NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

**INTERACTING THROUGH ARTEFACTS:
INTERACTION DESIGNERS AND SOFTWARE DEVELOPERS
ENGAGING IN THE ACTIVITY OF SOFTWARE CREATION**

by

Judith M. Brown
B.A. (hons) Psychology and Statistics, M.Sc. Computer Science
Queens University, Canada

A thesis submitted to
the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Psychology

Carleton University
Ottawa, Canada

©2010 Judith M. Brown

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-81567-0
Our file *Notre référence*
ISBN: 978-0-494-81567-0

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

There is little understanding regarding the collaboration of interaction designers and software developers on multidisciplinary teams creating real and novel software. This research examined the work of designers and developers in the early stages of this work. It focused on the talk- and artefact-mediated interactions that ubiquitously support designer-developer interactions. Cultural-historical psychology, and more specifically activity theory, provided the orientation and conceptual framework. Field data, including observational notes, interviews, photos, and meeting videos, were collected from eight organizations. These were analyzed using activity systems analysis (Engeström, 1986), interaction analysis (Jordon, 1995), and grounded theory (Strauss & Corbin, 1998). This research was conducted from four perspectives. From a historical perspective, the results illuminated the challenges presented by the amorphous nature of the software object, the continuous evolution of that object over decades, and the contradictions inherent in the roles of designers and developers that reflect the challenges of work within this domain. From a cultural perspective, the results revealed that software-creation work is really a system of activity systems, and that making strong connections between activity systems, using various means, including bridging artefacts, is important to achieving the varied social and cultural aims for the software. From a personal and situated perspective, the results showed that designers and developers enact Interactional Identities that determine their approach to the use of artefacts within the team, and that Interactional Identities serve to fill an important gap concerning how to fulfill the role of being a designer or developer. More experienced designers and developers enacted Interactional Identities that showed they more fully appreciated the collaborative nature of their work. From an artefact-centric perspective, the results revealed classes of meeting artefacts that support designer-developer interactions including interface proxies, computational proxies, stories, and models. Meetings were initiated to address product or project tensions, but broadened in scope and had phases including presenting, simulating, enacting, or exploring. Overall, the studies contribute a richer understanding of this domain, and revealed the social systems and individual strategies that shape software creation work and, in particular, the roles of artefacts in the joint work of designers and developers.

Acknowledgements

I would like to express my thanks to many groups and individuals who provided ongoing support. I am particularly indebted to the designers, developers, and managers who gave their support to my research because finding teams who would open their work lives to me was hard. I'm grateful to the eight project teams who said yes. Without their full support and encouragement this research would never have occurred. I am also thankful to my colleagues and the many researchers upon whose work this research is based. In particular, I am very grateful to the vigorous and energetic intellectual debate and the support that I received from the International Society of Cultural and Activity Theory (ISCAR), the individuals who participate in the Mind, Cultural and Activity (XMCA) mailing list, and the community of researchers at the Centre for Research on Activity, Development and Learning (CRADLE) in Helsinki.

I am also grateful to my co-supervisors, Prof. Robert Biddle and Prof. Gitte Lindgaard who provided me with guidance and support throughout the lengthy process of completing this dissertation. I was initially surprised, but then grateful, for the diverse perspectives and priorities my two supervisors imparted to my research. I am also thankful for my committee members for guidance: Dr. Connie Kristiansen in qualitative methods, Dr. Tzetanka Dobрева-Martinova in Russian psychology, and to my externals Dr. David Mould and Dr. Mary van der Riet for insightful comments and questions.

I am grateful to friends including my dissertation support group Neda, Cheshmak, and Devjani; supportive friends from the HOT lab and HOTSoft lab; and also my understanding friends Dawn, Cindy, Hai Toh, Susan, Marie, Shelley, and their partners.

I am indebted to support from family: my supportive partner and technical guru Jonathan Segel; our clever and kind sons Asher, Ben, and Remy; my siblings, Gord, Cathy (who was also my able transcriptionist), and Laurie; my family through various marriages including Stan, Honna, Lisa, Randy and Marc; supportive aunts, uncles, and cousins; and my Quaker family, particularly Signy, Steve, Erica, Peter, Jesse, and Beth.

To close, I'd like to dedicate my work to three women who each played an important role in my life and who passed away during the time I was working on my doctorate: my mother, Laurette Samson, my mother-in-law, Ellie Segel, and an exemplary person and friend, Rose Mae Harkness.

Table of Contents

List of Tables	vii
List of Figures.....	ix
List of Appendices.....	xii
Introduction.....	1
Motivation and Aim of the Research.....	1
Significance of Cultural-Historical Psychology	7
Influential Aspects of Cultural-Historical Psychology	9
Artefacts, Mediation, and Interaction	11
Activity as Context for Designer-Developer Actions	16
Analytical Methods in Cultural-Historical Psychology.....	19
Artefact Use in Software-Creation Work	24
Research Questions.....	29
Methods.....	31
Analytical Methods Used.....	31
Activity Systems Analysis	32
Contradiction Analysis.....	34
Interaction Analysis	36
Grounded Theory	40
Case Studies	42
Participants in General.....	42
Apparatus	43
Procedure in General	45
Perspective 1: Designer-Developer Interactions Understood Historically	47
Perspective 2: Designer-Developer Interactions and Activity.....	48
Procedure for Observing Software Work	48
Summary	50
Perspective 3: The Meaning of Interactions to Designers and Developers	51
Interviews and Interview Scripts	51
Procedure for Interviewing Participants	52
Summary	53
Perspective 4: Designer-Developer Interactions in Meetings.....	54
Procedure for Video Recording Meetings	54
Study 1: Mediating Artefacts	55
Study 2: The Development of the Activity from Meeting-to-Meeting.....	55
Study 3: Creative Work Within Meetings	55
Summary	56
Specific Organizations, Participants, and Software Projects.....	57
Personal Bias.....	64

Results and Discussion Perspective 1: Mediated Interactions and Historicity	66
The Nature of the Software Object	68
Disciplines Addressing Programs and Programming	73
The Nature of Software Development Work	73
Experts in Software Development Work	84
Contradictions in the Developer's Role	90
Disciplines Addressing User Interfaces and Designing.....	96
The Nature of Design Work.....	96
Experts in Design Work.....	109
Contradictions in the Designer's Role	115
Approaches to the Integration of Design and Development Work.....	117
Integration of Designer and Developer Processes	120
Integration Through Linked Designer-Developer Artefacts	122
Integration in Practice	127
Summary	130
Concluding Statements	132
Results and Discussion Perspective 2: Mediated Interactions and Activity.....	136
Activities at Talia Established to Create the Config-it Front End	138
Creating the Config-it Front End at Talia	141
Managing the Config-it Front End Project at Talia	170
Summary of Activity Systems Analysis at Talia	176
Interrelated Activity Systems at Jaba, Talia, Quadrow and Telespace.....	177
Focus-On Charts	183
Focus on the Talia activity Creating the Config-it Front End.....	188
Focus on the Jaba activity Creating and Maintaining PlayLearn-it games.....	196
Focus on the Telespace activity Creating the Sell-it Front End.....	204
Focus on the Quadrow Activity Creating Calcul-it	212
Summary	216
Concluding Statements	218
Results and Discussion Perspective 3: The Meaning of Mediated Interactions.....	222
Approach to the Analysis.....	222
Grounded Theory Adapted for Cultural-Historical Psychology	222
How Designers and Developers See Their Work	230
Full Case Study: Designer Owen's Interactional Identity	230
Full Case Study: Developer Wil's Interactional Identity.....	241
Abbreviated Case Studies	250
The Interactional Identities Model.....	253
Actual Work and the Interactional Identity.....	256
Discipline Specified Roles and Interactional Identities	256
Roles as Understood in CHAT and Interactional Identities.....	258
Individuals and Interactional Identities.....	258
The Relationship Between the Designer and Developer's Interactional Identities	259
Projects and Interactional Identities.....	260
Organizations and Interactional Identities	260

Team Work and Interactional Identities.....	261
Interactional Identities Over Time.....	262
Interactional Identities and Their Development Over a Career	262
Development of the Key Components of an Interactional Identity Over Time.....	266
Outside the Interactional Identity Model	276
Concluding Statements	280
Interactional Identity and the Meaning of Artefacts	283
Results and Discussion Perspective 4: Mediated Interactions and Meetings.....	286
Study 1: Shared Designer-Developer Artefacts.....	286
The Development of a Categorization of Artefacts	287
Meetings as Artefacts.....	289
Artefacts Used in Meetings of Designers and Developers	293
Summary	312
Study 2: Development of Creative Work from Meeting to Meeting.....	313
An Analysis of Jaba Meetings	314
Summary	330
Study 3: Creative Work within Meetings	331
Jaba Meeting Case Study	333
Interpretation and Significance of the Jaba Case Study.....	354
Analysis of Meeting Phases at Jaba, Talia, Quadrow, and Telespace	357
A Model of Meeting Activity	360
Summary	363
Overall Discussion and Conclusions	365
Criteria for Constructivist, Qualitative Research and an Evaluation.....	365
Social Validity	365
Subjectivity and Reflexivity	366
Adequacy of the Data.....	367
Adequacy of the Analysis	370
Consequential Validity.....	372
Validity and Reliability Revisited.....	373
The Four Perspectives and Their Research Questions: An Assessment of Findings .	374
Historical Perspective	375
Cultural Perspective	377
Participant-Centric and Situated Perspective.....	379
Artefact-Centric Perspective	382
Implications of the Findings	386
Integration of the Results	386
Theoretical Contributions	388
Research Methods	394
Practical Implications.....	397
Deepened Understandings	399
Future Research and Closing Remarks	401
References	403

List of Tables

Table 1 <i>Perspective 1-Designer-Developer Interactions Understood Historically</i>	47
Table 2 <i>Perspective 2-Artifact-Mediated Interactions and Activity</i>	51
Table 3 <i>Perspective 3-The Meaning of Interactions to Designers and Developers</i>	54
Table 4 <i>Perspective 4-Meetings Involving Designers and Developers</i>	57
Table 5 <i>Characteristics of the Participating Organizations</i>	58
Table 6 <i>Characteristics of the Software Being Created in the Organizations</i>	59
Table 7 <i>Characteristics of the Designers</i>	61
Table 8 <i>Characteristics of the Developer</i>	63
Table 9 <i>Contradictions in the Developer's Role</i>	92
Table 10 <i>Contradictions in the Designer's Role</i>	116
Table 11 <i>Observational Data</i>	136
Table 12 <i>Some Aims for the Config-it Project by Perceived Priority</i>	167
Table 13 <i>Product Creation and Project Management Attributes Compared</i>	179
Table 14 <i>Activities at Talia</i>	182
Table 15 <i>Key Concepts Contributing to the Category Interactional Identity</i>	226
Table 16 <i>Interactional Identities Grouped by Specialist and Experience</i>	251
Table 17 <i>Interactional Identities—Handles and Descriptive Phrases</i>	252
Table 18 <i>More- and Less-experienced Designers' Goals</i>	267
Table 19 <i>More- and Less-experienced Developers' Goals</i>	268
Table 20 <i>More- and Less-experienced Designers' Views of the User Interface</i>	270
Table 21 <i>More- and Less-experienced Developers' Views of the User Interface</i>	271
Table 22 <i>More- and Less-experienced Designers' Views of Artefacts</i>	273
Table 23 <i>More- and Less-experienced Developers' Views of Artefacts</i>	274
Table 24. <i>More- and Less-experienced Designers' Views of Contradictions</i>	275

Table 25 <i>More- and Less-experienced Developers' Views of Contradictions</i>	276
Table 26 <i>Developers' Views of Designers</i>	278
Table 27 <i>Designers' Views of Developers</i>	279
Table 28 <i>Meetings at Talia, Jaba, Telespace, Quadrow</i>	291
Table 29 <i>Brief Description of Artefacts Used in Meetings</i>	294
Table 30 <i>Meetings at Jaba</i>	329
Table 31 <i>Meetings Involving Designers and Developers and Number Video Recorded</i>	331
Table 32 <i>Patterns in Talk- and Artefact-Use in a Design Meeting at Jaba</i>	341
Table 33 <i>Value of Sketches in Interactions Below Level of Conscious Awareness</i>	352
Table 34 <i>Roles of Stories in Interactions Below the Level of Conscious Awareness</i>	353
Table 35 <i>Roles of Stories and Sketches Together in Interactions</i>	353
Table 36 <i>Role of Lists in Interactions</i>	353
Table 37 <i>Counts of Key Meeting Phases by Meetings</i>	358

List of Figures

Figure 1. Engeström’s activity system triangle.....	34
Figure 2. Creating the Config-it Front End activity.....	169
Figure 3. Managing the Config-it Front End Project activity.....	175
Figure 4. Focus-On Chart for the Config-it Front End at Talia. Shown is the Creating the Config-it Front End activity in relation to other relevant activities in this multi-national organization.....	195
Figure 5. Focus-On chart for the PlayLearn-it game at Jaba. Shown is the Creating and Maintaining PlayLearn-it activity in relation to other relevant activities at this fully-funded venture company.....	203
Figure 6. Focus-On chart for the Sell-it Front End at Telespace. Shown is the Creating the Sell-it Front End activity in relation to other activities in this service organization.	211
Figure 7. Focus-On chart for Calcul-it at Quadrow. Shown is the Creating Calcul-it activity in relation to other activities in this well-established organization.....	215
Figure 8. A model of designer Owen’s Interactional Identity movie director.....	239
Figure 9. A model of developer Wil’s Interactional Identity, binder.	249
Figure 10. The Interactional Identity model.	255
Figure 11. A Model of designers’ Interactional Identities based on years of experience.	264
Figure 12. A Model of developers’ Interactional Identities based on years of experience.	264
Figure 13. Example of software under development. Jaba’s game prototypes were so realistic it was treated as if it was the software under development. This particular prototype had placeholder elements for graphics, art, and content, but was still a playable game. Gamers could interact with the elements, select characters and perform actions.	295
Figure 14. An interface proxy. An interactive prototype of a gaming element being designed at Jaba.	298
Figure 15. An interface proxy. A prototype developed by a contractor to show the visual aspects of an interface at Quadrow. This design constrained the look of the Calcul-it software.....	298
Figure 16. An interface proxy. A whiteboard sketch of two connecting screens drawn by the designer at Quadrow to show some new behaviour in the user interface.	299

Figure 17. An interface proxy. A prototype developed by the designer at Telespace using Powerpoint. The prototype shows the screens to be developed and the flow of interaction between the screens. This prototype was mostly used in its paper form, and was carried from desk to desk for chats between any of the members of the software team.	299
Figure 18. An interface proxy. A drawing created by Talia's designer. Typically the designer would create two or three prototypes for each meeting. Each prototype presented a different concept or idea.....	299
Figure 19. A computational proxy. This table drawn on a whiteboard was created at Quadrow to simulate a computation. The computational engine at Quadrow was very complex. Work with this artefact addressed the scope of the computation, and therefore the scope of the application. Quadrow also had a spreadsheet programmed to simulate their computational engine.....	302
Figure 20. A computational proxy. This drawing was created at Jaba, with paper, pencil and ruler. The team played this game with a penny and dice. The purpose was to explore the variables in the game and to determine how the variables should be used together.	302
Figure 21. A project plan. A traditional project plan used at Telespace. Tasks for developers, designers and designopers are in rows, time (days) are in columns. Data in the table included planned start and completion dates, current status, resource use, and current activities.....	303
Figure 22. A project plan. This drawing served as plan at Telespace. Each little box represented a feature; the designers tracked how many of these were completed using this diagram.	304
Figure 23. A model. A model of Bloom's taxonomy that supported the Jaba content/learning designer's thinking about learning in the game.....	307
Figure 24. A model. A model of flow that supported the Jaba content designer's thinking about the gamer's experience. In the designer's opinion, gamers should be "aroused, in a state of flow, or in control." Any other state would be undesirable.	308
Figure 25. The sketchbook prototype used in Meeting 1.....	317
Figure 26. A plan for how the game's model will map to the three other game stages..	320
Figure 27. The developer's sketch (i.e., a specification) that shows how to organize the content for the game in an XML file.	321
Figure 28. A list of graphical elements (i.e., ideas) for the new game.	321
Figure 29. A list of the aspects of the Sketchbook prototype (i.e., the software under development) that should be retained (i.e., good ideas).	322

Figure 30. The first design for the action selector component (i.e., an interface proxy) is a table with buttons.....	323
Figure 31. Sketch of a grid (i.e., a computational proxy) that maps a person and an action to an event, which identifies a formula that changes the game's score.	325
Figure 32. A tech tree (i.e., an exemplar) from the game of Civilization.....	326
Figure 33. A sketch of what the tech tree would look like in the game (i.e., an interface proxy).....	326
Figure 34. A sketch that captured a designer's idea about game actors.	335
Figure 35. A list that captured a designer's idea about game icons.....	336
Figure 36. Two hundred and nine artefacts used in the Jaba meeting by category.	340
Figure 37. Each phase is centered around a set of artefacts. The artefact types within a phase are similar types for phases of the same type.	340
Figure 38. Counts of words, stories and visuals for various phases.	345
Figure 39. Phases and subphases in meetings.....	345
Figure 40. Proposing and Evaluating subphases and Assessing phases ordered temporally.	346
Figure 41. Results of analysis of contradictions, a 12-way contradiction. The team aimed to achieve all 12 attributes (i.e., objectives) within the one game; the challenge for the team was to work out how to do this. At times when attributes were in conflict, occasional tradeoffs were necessary. The visualization shows the many attributes that required consideration. Some, but not all, were in conflict with each other.....	350
Figure 42. Phases in meetings involving designers and developers. The transformational work of developing the object of the activity occurs in presenting, simulating, exploring, and enacting phases.....	361
Figure 43. A model for the activity in meetings involving designers and developers. Seven meeting phases are shown. There is no ordering of the phases within the circle. Meetings may wind down and resume. Phases on the left side of the circle are project-focused, those on the right are focused on the interface object.	362
Figure 44. A model for meetings of designers and developers that shows the typical types of artefacts used in various phases.....	363

List of Appendices

Appendix A: Glossary of Terms	422
Appendix B: Demographics questions.....	426
Appendix C: Interview questions—Artefacts used in software-creation work	427
Appendix D: Interview questions—Software-creation work over time	429
Appendix E: Applying cultural-historically influenced grounded theory	430
Appendix F: Nineteen Further Case Studies of Interactional Identities	431
Appendix G: Tables of Meetings at Talia, Telespace, and Quadrow	475
Appendix H: Details of meetings that were video recorded	482
Appendix I: Evidence from this research supporting the existence of contradictions in the developers' roles	484
Appendix J: Evidence from this research supporting the existence of contradictions in the designers' roles	485

Introduction

Motivation and Aim of the Research

Software plays a large and increasing role in modern economies and in everyday life, but the collaborative aspects of how that software is created are poorly understood. In particular, the collaboration between interaction designers and software developers has been seldom addressed. Further, there are not many real-world studies that have been designed to carefully collect data from this type of work situation and rigorously analyze it using a solid theoretical foundation. As a consequence, little understanding of the interactions between the members of the multidisciplinary teams who regularly create software exists; this is particularly the case for interdisciplinary interactions.

This dissertation consists of a series of studies of the work of software designers and developers who are examined in the process of creating novel and complex software products. The research examined the interactions between developers and designers, both of whom hold important roles in creating software intended for users.

The focus of this work is on the joint activity of developers and designers in the early stages of creating complex end-user software products, sometimes known as *concept design*¹. In this stage of the work, the software product proceeds from a vague idea to a well-developed one with important elements more or less fully specified and possibly even implemented, and with both designers and developers necessarily involved in this work. At the end of this stage, the product team knows, in much more concrete terms, what product it is developing and they have usually begun the development work.

¹ Italicized terms are listed and defined in a glossary of terms found in Appendix 1 of this document.

This stage of work is highly creative and messy and inherently difficult for many reasons: the rapidly changing real-world requirements, the unusual nature of software products, and the multidisciplinary nature of software-creation work. Other members of the team typically include business representatives, managers, testers and writers. Artefacts, such as the documents, mock-ups, or drawings of software objects that team members produce and use to interact with one another, figure prominently.

An examination of the role that artefacts play in the interactions between designers and developers helped to shed light on this important creative activity and those engaged in it. The purpose of exploring this activity was to provide insight into the nature of creativity in software teams and to understand how designers and developers manage to accomplish this challenging work, and especially how artefacts support their interactions.

To achieve these aims and examine designer-developer interactions and software creation in all of its complexity, this research was conducted in real workplaces. When working on site, first a designer, and then a developer they worked with, was shadowed and then interviewed. When shadowing was not possible, designers and developers were simply interviewed on site. To ensure that generalizations could be made at the end of this research, this work was conducted in eight organizations.

Designers and developers are the members on the team whose direct actions create the executable software. Developers create the executable software by writing code and compiling it, but today's designers can also create executable software. Sophisticated tools that allow designers to build the user interface through drag and drop techniques are also capable of converting the built user interface into executable software. The developer can then combine the designer's executable software with his or her own to

produce the software in its entirety. Because of these abilities, the role of the designer and developer are of critical importance when investigating how software is created. About half of the designers in this research used tools that allowed them to create their part of the executable software. For the rest, the developers reproduced the designers' work and then incorporated it with their code to create the executable software.

Points of interaction, rather than solitary work, were a focus for this research because it seemed that the work changed or shifted more dramatically at times of interaction. However, this does not imply that solitary work, also observed in this research, is unimportant or unproductive, it merely suggests that solitary work proceeds more smoothly.

The software developers in this research were specialists trained to create reliable, industry-standard software applications. All were responsible for writing code to implement the appearance and behaviour of the user interface and therefore worked closely with designers. However, most also had other responsibilities, including writing the program's logic, determining and arranging the software's components, writing code to connect to other software, or writing code to manage external data.

Designers in this research were individuals who were specifically responsible for both the visual and behavioural aspects of the user interface. These designers referred to themselves as human-computer interaction specialists, user interface designers, user experience designers, human factors specialists, graphic designers, content designers, or learning designers.

Excluded from this research were artists and some graphic designers whose work solely addressed the appearance of the software because this type of designer has relatively little interaction with developers as their work does not overlap. Designers who

did not work with developers, such as many Web designers, were also not included in this research. In this case, sometimes the development work is outsourced and is nonnegotiable from the developer's perspective, and in other cases development work is fully automated. Similarly, developers who did not work with designers—even developers who designed user interfaces—were excluded. In these cases, designer-developer interactions are also not an issue because such projects have no specialist invested principally in the software's interface. In all of these cases, the tension between what the designer of the user interface wishes the user interface to be and what it is possible to create is not negotiated. The Web designer is limited by their tools' abilities. The developer who is also a designer is fully knowledgeable about the limitations of software and designs with this knowledge in mind.

The work of designers and developers is artefact-rich. Because developers cannot work directly with executable programs, they work indirectly through representations of the software such as program code (the code is not the software), abstract representations of software parts, diagrams that show the dynamics of the software, and other representations. Designers must also work indirectly and use many varied artefacts in their work, such as drawings of the user-interface-to-be, scenarios, prototypes or wire-frame diagrams. In either case, because software products are complex, no one representation captures all aspects; hence each specialist has many representations with which to work. Still, one might expect that because each specialist works on the software indirectly, they need not work together extensively. However, designers and developers do come together to expose and realign their work to proceed efficiently and effectively. This involves much communication through talk and artefact use.

While designers and developers work both separately and together, the focus in this dissertation is on the times when they meet because at these times the individual work of each specialist may be redirected as they and their software team jointly address a multiplicity of aims that sometimes conflict. While these interactions may be mediated through e-mail or video-conferencing technology, this research only studied designers and developers working face-to-face. This provided the greatest potential for a rich set of communication techniques to be employed and the opportunity for unconstrained use of artefacts. Face-to-face interactions provide the best opportunity for studying interactions through artefacts. In face-to-face interactions, artefact use is naturally combined with other means of communication to achieve a range of goals that interacting individuals may have, such as developing an idea, remembering the status of their work, or addressing a problem. In the same way that linguist Charles Fillmore said, “the language of face-to-face conversation is the basic and primary use of language, all others being best described in terms of their manner of derivation from that base” (Clark, 1996, p. 8), the viewpoint taken in this work is that the study of face-to-face designer-developer interactions is a way of accessing and understanding the most basic elements of designer-developer interactions.

Other aspects of this research have not been constrained. For instance, developers may be engaged in any number of approaches used to develop software, such as waterfall-like processes, where tasks are organised in a sequential fashion, or Agile practices (Beck et al., 2001; Cockburn, 2002), where work is organized so that the software is produced collaboratively in carefully considered increments. This research aimed to include a variety of approaches to software development work.

Developers in this study were a fairly homogenous group relative to designers, as most were computer scientists. However, designers varied in their education, training, and their view of their role. Some understood their role as designing the best possible user interface for their users, others saw it as designing the interaction between a person and a piece of technology. Some designers felt they had more to say about the entire shape of the product and others were focused on the content (i.e., the dialogue sequences for characters in games, or the selection and structuring of information in an interactive, Web-delivered course).

To study purposeful designer-developer interactions and to study these over time, each of the eight case studies included in this research focused on the creation of a single software product and a designer-developer pair that worked together on that product. The type of software product was not constrained and varied from Web-based products to custom-built in-house applications. All were real software products intended for use by anywhere from a hundred to many thousands of users and, although the research did not follow the work to the end, all were successfully completed and seven out of the eight products were released and are now in use.

The aim of the dissertation was to understand how artefacts mediate the interactions of designers and developers engaged in the early stages of software-creation work by observing this work as it unfolds through talk- and artefact-mediated interactions. The work comprised a series of studies. The studies were used to reveal patterns of interactions and artefact usage, which were found to be common in a wide variety of software-creation work, and from these patterns, make inferences about the minds of designers and developers in action. The work of creating software was explored using cultural-historical psychology. This psychology has a rich perspective on artefacts

and artefact use, and these perspectives figure prominently in its aim to deepen the understanding of the human mind in action. Cultural-historical psychology is applied here to better understand certain aspects of software-creation work. Therefore, approaches to data collection and analytical techniques consistent with cultural-historical psychology are used and have been developed as a part of this dissertation.

Significance of Cultural-Historical Psychology

Cultural-historical psychology was chosen as the framework for this dissertation. This psychology is seen in this research as one of a number of psychologies in the broader field of psychology, a viewpoint that was also expressed by Lev Vygotsky (1994). At the same time, cultural-historical psychology is also understood in this dissertation as a dynamic and diffuse framework that has been variously interpreted by researchers and practitioners (Daniels, 2008b, Ch. 1). This section describes the aspects of cultural-historical psychology that are leveraged here, the underlying assumptions of this psychology, its views on artefacts as mediators, and the role of activities as a “context” for mediated actions and interactions. Relevant concepts are discussed briefly, followed by an explanation of their relevance to this research.

Cultural-historical psychology (Cole, 1997; Ratner, 1991; Wertsch, 1985) was founded by Vygotsky (1896-1934) (1978b; 1986) and emerged in the early days of the Soviet Union, around 1920 (Glenn & Jesmaridian, 1999). Vygotsky’s psychology was a reaction to the inadequacies of other Russian and Western psychologies at the time, including Russian reflexology, American behaviourism, Piagetian psychology, introspective psychologies, Gestalt psychology, any psychologies that were basically a classification system, and psychologies whose main focus was the development of metrics (Leontiev, 1997; Vygotsky, 1994; 1997). Cultural-historical psychology drew

attention to human behaviours as a means to understand the mind in action, drawing heavily on the writings of German philosophers, particularly Hegel and Marx. In cultural-historical psychology, the mind or psyche is seen as a dynamic system that develops through lived, cultural experiences. Cultural-historical psychology was meant to be both a unifying psychology and a practical psychology. Its name appropriately describes a psychology that emphasizes the dominant role of culture in human internal and external actions and the value of employing a historical or process-oriented approach to studying the human psyche. Vygotsky's colleagues, Luria and Leontiev, developed his psychology further, as did many modern counterparts including Harry Daniels (2008b), Yrjö Engeström (2008), Michael Cole (1997), and James Wertsch (1998). Especially important here is the work of researchers who further developed the ideas of mediated action, activity, and culture—those working in the area of cultural-historical activity theory (CHAT). CHAT grew out of several roots, but most particularly cultural-historical psychology to which it still has close ties, and broadened into a multidisciplinary framework (not a theory) used by psychologists and researchers from a host of other disciplines (van der Riet, 2009).

Vygotsky's premise was that the mind or psyche develops through lived, cultural experiences. In this dissertation, to study designer-developer interactions mediated by culture, designers and developers were studied at work in real organizations. Notes taken during observation periods, video recordings, and photography were used to capture designers and developers while they worked, both separately and, more importantly, together. Observations in the field provide insight into the nature of software-creation work and designer and developer minds in action.

Influential Aspects of Cultural-Historical Psychology

Cultural-historical psychology takes a dialectical approach, which has a number of implications. Firstly, a dialectical approach looks for general and abstract laws that capture the essential qualities or characteristics of the development of nature, society, and thought. This principle directed this work in finding patterns in designer-developer, artefact-mediated, interactions that reveal something about how designers and developers engage in the process of software creation.

Secondly, a dialectical approach sees the world in a state of continuous flux. In this dissertation, this approach was used to examine software-creation work and designer-developer interactions from a developmental perspective that includes a consideration of change over time.

Thirdly, a dialectical approach considers that contradictions are omnipresent in all aspects of life. Vygotsky, for instance, saw the world in a constant state of flux and believed that everything held internal dialectical contradictions, as does Engeström (1987) whose focus is on studying contradictions and their role in group activities. Contradictions are forces or persistent tensions—not opposites, negations, breakdowns, disturbances, or conflicts—that need not necessarily be distinct, but that draw an activity in one direction or another. Contradictions can be two-way forces, or multiway tensions, and are ubiquitous and plentiful in all activities including software development work. Despite the negative connotation of the word “contradiction”, a contradiction is neither inherently good nor bad. In fact, many contradictions described in this dissertation were not experienced as negative events and were good in that their resolution resulted in the creation of a better software product. In this dissertation contradictions were revealed

through a literature review and from observations and videos of designers and developers interacting.

Fourthly, a dialectical perspective considers that change is due to the resolution of contradictions. For example, suppose a software product needs to be both aesthetically appealing and responsive. The process to determine how this can be achieved is a difficult and common contradiction that developers and designers resolve. This research is sensitive to both contradictions that influence designer-developer interactions and contradictions that are resolved through designer-developer interactions.

A fifth consideration of a dialectical approach is that elements in a system are interdependent. Using activity-systems analysis (Engeström, 2000), systems of interdependent elements in software-creation work that include the designer, developer, shared artefacts and other elements are identified. These systems provide the natural context for the study of designer-developer interactions. In this research, both the broader activity and designer-developer interactions are kept in mind and one is not ignored at the expense of the other.

A final consideration of a dialectical approach is that change can occur abruptly. By focusing on the times when designers and developers meet, rather than when they work independently, this research was able to capture moments where change is more abrupt.

Materialism is also important in this research, as in Vygotsky's work and the work of other cultural-historical psychologists. A materialistic viewpoint focuses on what is perceived through the senses and the result is that material aspects of the world, such as artefacts, figure prominently. Here, the artefact-mediated interactions of designers and

developers are of prime importance and this research is used to understand the role these artefacts play by conducting studies in the workplace.

Artefacts, Mediation, and Interaction

Artefacts, which figure prominently in designer-developer interactions, are understood here, as in Cole's (1997) work, where he stated,

an *artifact* is an aspect of the material world that has been modified over the history of its incorporation in goal directed human action. By virtue of the changes wrought in the process of their creation and use, artifacts are simultaneously ideal (conceptual) and material. ... They are manufactured in the process of goal directed human actions. They are ideal in that their material form has been shaped by their participation in the interactions of which they were previously a part and which they mediate in the present. ... Defined in this manner, the properties of artifacts apply with equal force whether one is considering language/speech or the more usually noted forms of artifacts such as tables and knives which constitute material culture. What differentiates the word 'table' from an actual table is the relative prominence of their material and ideal aspects and the kinds of coordinations they afford. No word exists apart from its material instantiation (as a configuration of sound waves, hand movements, writing, or neuronal activity), whereas every table embodies an order imposed by thinking human beings. (p. 117)

Cole's cultural-historical perspective draws attention to several aspects of artefacts, including their material nature, their ideal nature, the history of their formation, and their purpose from both a psychological and social perspective. Artefacts of interest in this dissertation were those that were purposefully designed to support the work of software creation and that mediated designer-developer interactions. Many artefacts in the workplace of designers and developers were of no interest, such as mugs or tables and chairs, unless these had been arranged in a unique way to specifically support the work of creating software. The material aspect of artefacts found in designer-developer interactions varied greatly, from representations on computer screens, to paper-based products, to functional prototypes of envisioned real-world products, to artefacts found in

the talk of the designers such as references to models. The ideal aspect of artefacts also varied greatly in the range of meanings attributed to artefacts by their creators and consumers. Both the material and ideal aspects of artefacts are of relevance in this work.

In the course of this research the understanding of artefact provided by Cole was extended as a result of studying software creation activity. For example, I “observed” amorphous artefacts mediating designer-developer interactions, such as software frameworks, a digital artefact that cannot be seen, heard, or touched and is therefore not material, but is nevertheless real. Also challenging to Cole’s definition were artefacts such as interactive prototypes that are interactive digital mockups of the software-to-be that are experienced differentially over time by those who explore them through their software interface.

Although artefacts are of great importance here, this is not a dissertation about artefacts, but about artefacts as mediators of meaningful interactions. Vygotsky (1978b; 1986) studied mediated actions, where a subject or person interprets the world and acts in a purposeful, goal-oriented way to transform it by using a tool (e.g., a hammer) to change an object in the world (e.g., nails and bits of wood to be transformed into a table). The *object* of an action is the element or elements that are being transformed by that action. The term object is used extensively in this research to identify what actions are directed toward, so it is a more specific definition of object than the one in common usage. A term that is very closely related to object is *objectives*, which loosely means the desired attributes, which are understood as the social and cultural properties of the object under transformation (Kaptelinin, 1996; 2005).

In cultural-historical psychology subject, tool, object, goal, and action are interdependent and seen as a single unit. There is no division between the world and the

acting subject. This explains the existence of phrases such as the “mind in action” (Wertsch, 1998) or “thinking as action” (Santamaria, Cubero, & de la Mata, M. L., 2010). Vygotsky (1978a) identified two types of material mediators: tools that mediate external actions (e.g., hammers that support the action of joining wood for a carpenter) and signs that mediate internal actions (such as bookmarks that support remembering for a reader), suggesting these should be studied together, but also noting that there were other types of mediators. In workplaces where software products are being created, the concept of mediated action applies. Designer and developers regularly use tools, the former drawing tools and the latter coding tools, to produce useful artefacts such as drawings or interactive prototypes. There are also many examples of signs such as to-do lists that mediate remembering and self-regulation, and other artefacts that embody aspects of both tools and signs, such as annotated documents.

In a similar way, the mediated interactions studied in this research, which comprise acting individuals and mediating artefacts, are seen as purposeful, mediated units. A simple example of such an interaction is a handshake between two people. Here, the interacting individuals use gestures and words to jointly achieve a culturally appropriate greeting. In a similar way, designers and developers interact to achieve specific purposes and use talk and artefacts as mediators. These artefacts are usually the objects of previous tool-mediated actions conducted during solitary work. Here it is posited that there is an object of transformation in interactions in general (meaning the interactions are targeted toward transforming something), and in designer-developer interactions specifically. As in the case of mediated-actions, this object is also considered to be an artefact. For instance, the designer and developer may jointly transform a

drawing of a software product through their actions, by making notes on it or embellishing it by sketching.

This discussion of artefacts as mediators also shows the futility of unequivocally categorizing an artefact as a tool, sign, or object as the role an artefact plays is determined by the actions or interactions in which it is used to act or interact. For this reason, during analysis, photos of artefacts were embedded in detailed research notes and video images were embedded in transcripts of videos so that they would always be considered in the context in which they were used. This research categorized artefacts that appear in mediated interactions with careful attention to the purpose of the interactions in which the artefacts were embedded.

A further assumption of many cultural-historical psychologists is that individual action and individual or group activity must be studied simultaneously or individual action cannot be understood (Leontiev, 1978c). The thrust of the argument is that motivation (e.g., to build a table) associated with human needs (e.g., my family needs a place to eat) is the reason for the activity and not the reason for specific actions (e.g., the more specific reason given for hammering is to join two pieces of wood). Hence, an analysis of activity accesses deeper reasons for an individual's actions than an observation of the completion of an action or a series of actions, or the satisfaction of a short-term goal or a series of short-term goals can imply. A second reason is that actions and activities constitute each other, meaning that without actions there is no activity, and without an activity there are no actions.

Thus, designer-developer interactions and software-creation activity involving specialists from other disciplines as well are studied simultaneously. While this research is about designer-developer interactions, the interactions cannot be understood apart from

software-creation work because they are a part of one another, and neither can be fully understood in isolation.

Vygotsky studied how individuals are shaped by mediators/artefacts, often referred to as a dual view of mediators because the previous view of artefacts as mediators emphasized how they were used to transform objects in the world. In contrast, Vygotsky showed how thinking is shaped by mediators such as language and culture. From this perspective, mediators are not seen as agents for actions, but rather as cultural legacies received from others who came before and who shaped the environment in which actions and interactions occur. Within teams that create software products, the cultural legacy consists of the tools, representations, processes, roles, etc. The designers, developers, and other team members inherit from the many people who previously developed and shaped these aspects of the environment. In software, this legacy has an impact on how software problems are viewed and solved and what one specialist expects of another in terms of work products. Because this dual nature of artefacts is so important, this dissertation begins with a historical review that sheds some light on designer-developer artefact-mediated interactions, thereby positioning this research within a historical context.

While Vygotsky saw culture (understood in a more traditional sense, such as Spanish culture) as a mediator, here *culture* is understood as a “system of artefacts” that “comes into being whenever people engage in joint activity over a period of time” (Cole, Levitin, & Luria, 2006, p. 301). Although many cultures exist simultaneously and are interwoven, the relevant culture that is stressed here is the culture that has been developed around the work of creating software. Similarly, adhering to Cole’s generalized understandings, mind is understood as the process of mediating behaviour

through artefacts in relation to a supra-individual “envelope” with respect to which object/environment, text/context are defined (Cole et al., 2006, p. 143). In studying designers and developers, the focus is on developing an understanding of the mind engaged in actions and interactions using culturally-imbued artefacts in relation to an envelope which is software creation activity.

Vygotsky always took a historical or genetic approach. A historical or genetic approach develops an appreciation of the origins and development of practice and the reasons for changes in practice. Such an approach aims for an explanation of practice versus a description of practice and reveals the cultural legacy embedded within tools (Vygotsky, 1978c). Two assumptions of the genetic approach are (a) that current practice is shaped and limited by previous forms of the current activity and that those engaged in the activity are largely unaware of the cultural legacy of their environment, and (b) that a genetic perspective provides explanatory power through the identification of forces that shaped past changes.

Activity as Context for Designer-Developer Actions

Leontiev (1972, 1978c), a colleague of Vygotsky, developed important key concepts introduced by Vygotsky. Leontiev (1978c) believed that “psychology, like every other science, cannot develop only as a sum of empirical facts. It cannot escape theory, and the whole problem lies in what theory will guide it” (p. 28). To Leontiev, the theoretical core of psychology was activity, and psychology was the study of “activity, its construction, its development, and its transformations ... its various types and forms” (p. 109) because “human life is activity” (p. 49). In describing activity, Leontiev suggested that it is “sensual,” meaning it involves the senses and is concrete and not abstract. The main feature of activity is the object of the activity, which directs it and links the activity

to human needs. The object of activity has the same sense as the object of an action, except that the object of transformation in an activity is a “higher-level” object transformed by many, possibly collective, actions. Other attributes of activities are their motives, form, methods, emotional intensity, time requirements, space requirements, and the physiological mechanisms of individuals engaged in activity. For the purposes of this dissertation, the concept of *activity* will be used to identify and understand the specific activity or activities in which designer-developer interactions are embedded. It will also be used to identify other relevant activities that interact with activities in which designers and developers are engaged. For software-creation work, the object of the activity is the thing the team is working together to create (e.g., the application, the feature, or the user interface).

To understand activity, Leontiev (1978e) introduced the concept of *reflection*, a human mechanism for perceiving and understanding the world (p. 6), and which is engaged when an individual perceives any aspect of the world, including artefacts. To Leontiev, reflection on reality, one aspect of which is individual consciousness, arises through activity. Activity theorists assume that psychic reflection (roughly one’s understanding of the world, and not anything to do with clairvoyance, telepathy, or the like) develops in parallel with activity. Hence, the link between the study of activity and the study of human reflexivity or psychic reflection is strong. This is reflected in Leontiev’s search for “units of analysis that carry psychological reflection and give rise to it or mediate it” (p. 6). Activity, he said, satisfied this condition. Put bluntly, Leontiev is saying that studying activity directly is a way to study the mind under development. Mind in action is studied directly through observations in this research, and psychic

reflection is studied indirectly through interviews conducted with designers and developers who were asked to reflect on their work and the roles that artefacts play in it.

Activity is most closely related to the aspect of psychic reflection called consciousness (Leontiev, 1978a), a higher-level form of the psyche (a lower level form is *unconsciousness*, which will be called nonconsciousness here to avoid confusion with the normal meaning of the term). The content of “individual consciousness” governs the awareness and execution of goal-oriented actions, such as conveying a design idea. The content of the nonconsciousness determines environmentally-, conditionally-determined actions called *operations* (or automatic actions) that are beneath the individual’s level of awareness. In this dissertation, when designers and developers were being observed, they were asked to verbalize their goals. This gave me access to their conscious goals and allowed me to distinguish them from operations. Examples of the goals they vocalized were “to design an icon for a new button” or “to verify the connection with the database.” Participants were also asked about their overall goals in interviews and here they gave more general responses such as “to ensure the interface is simple” or “to complete on time.” Goals tightly associated with small strings of actions were easy to surmise (I was never surprised by a participant’s stated goal), but those that influenced many actions were not easy to surmise, and therefore the interviews were very valuable.

Personality is understood similarly to consciousness in that it is an aspect of reflection. Leontiev (1978b) described personality as “a psychological neoformation that is formed in the life relations of the individual as a result of a transformation of his activity” (p. 105). “Man (sic) becomes a personality only as a subject of social relations” (p. 105) and personality is “the ripening of genotypic traits under the influence of the social environment” (p. 107). In activity theory, to understand personality is to

understand the individual's motives, their motive-structure and their connections with the world. The concept of personality became relevant in this research when interview transcripts were analyzed.

As an analytic technique for understanding the psyche, Leontiev developed a three-level schema that described the sphere of analysis of human activities and directed attention to the transformations between these three levels, which are analyzed separately and not as a hierarchy. The three levels are (a) motive and activity, (b) goal and action, and (c) "instrumental conditions" (i.e., environmental conditions) and operation. Leontiev asserted that human activity was driven by motives linked to needs, that individuals took conscious actions based on their personal goals, and that the human body subconsciously executed operations (i.e., subconscious actions) based on conditioned reflexes determined by a wide variety of environmental conditions. Leontiev's three-level schema is a valuable analytical tool and this research structures the analysis of videos of design meetings using this schema. However, unlike Leontiev's hierarchy, interacting individuals, their activities, joint actions, and joint operations are analyzed, rather than separate individuals and their activities, actions, and operations.

Analytical Methods in Cultural-Historical Psychology

Although much research in cultural-historical psychology has been conducted without specific analytical methods or tools, for this dissertation considerable effort and attention was put into setting the direction for analysis and selecting just the right tools. The data collected presented challenges for analysis and integration because it included notes, transcribed interviews, photos, and videos. There was also a very large amount of data to analyze. This required analytic tools with flexible units of analysis that could be adapted for different data types, yet also had the potential for generating theory or models

about software creativity. The specific analytic tools selected are described in the methods section.

Vygotsky, his students, and colleagues used the method of double stimulation to study human use of signs (Engeström, 2007). For example, Leontiev (1978d) used a set of cards as objects to be remembered (the first stimuli set) and another set (the second stimuli set) as potential mediators to study the development of memory in children. Luria used verbal instructions as mediators to study the development of regulation in children (Cole et al., 2006). In all cases, Vygotsky and his colleagues observed the behaviour of children at different ages accomplishing tasks they were given with and without auxiliary artefacts. Each study shed light on the role of artefact used in the development of specific psychic functions (i.e., mental abilities) and provided evidence that the use of artefacts to support psychological functions was internalized over time. However, this method, valuable for providing evidence for the premises of cultural-historical psychology and insight into the development of children, was not suitable for studies of adults engaged in creating software that were to be conducted in a workplace setting.

Vygotsky's general approach to analysis was influential. Vygotsky was interested in a variety of time frames, including evolutionary development, the development of culture or group activities, the physical and psychological development of the individual, and the development of individual actions and conditioned reflexes to environmental conditions (Wertsch, 1985). An important issue for Vygotsky was identifying the best unit of analysis with which to study changes over time. He sought units that would provide maximal insight into the human psyche, which was commonly "the artefact-mediated action of an individual." However, Vygotsky explored other units, arranged chronologically, that included "'an instrumental act,' 'the highest function of the psyche,'

‘meaning,’ ‘sense,’ and ‘experience’” (Yaroshevsky, 1996, p. 170). Vygotsky sought a unit that was indivisible and included aspects of both the individual and the social.

Meaning, for instance, encompassed both the individual and the social because only an individual can perceive the meaning of a sign, while the sign itself is socially constructed. My research also works from that perspective in that, throughout the analyses of the data, attention was given to identifying appropriate units of analysis for a process. Each perspective studies a process as it unfolds over time, with each analyzed in units appropriately selected to capture all the essential aspects of designer-developer artefact-mediated interactions.

For analyzing designer-developer interactions, both the talk between the specialists and their use of artefacts are analyzed. The researchers in cultural-historical psychology who provided a valuable orientation to understanding talk and artefact use are reviewed below.

Vygotsky (1986) hypothesized that individuals were wired for speech, that speaking abilities developed meaningfully as babies and children interacted with others, and that learning to speak was critical to the development of thinking skills. Language, he said, was internalized, and then transformed into thoughts. Talking, therefore, was insight into thinking.

Mercer and Staarman (2005) saw language as “not just a means by which individuals can formulate ideas and communicate them,” but a “means for transforming experience into cultural knowledge and understanding” and “a means for people to think and learn together” (Mercer, 1995, p. 4). This research saw speaking as a process that not only communicates ideas, but also transmits culture, and develops shared knowledge. However, this view is extended to artefact use as well because many artefacts, especially

in the software domain, are premised on a visual language that can be expressed and interpreted.

Daniels' (2005a; 2005b) view was that interpersonal relations are regulated by language and that language is produced through social relations. He argued that explanations of discourse should also consider the social position of "conversants" because a person's social position can influence who speaks, what is said, how utterances are formed, and so on. In this dissertation, the social position of conversants in software-creation work is recognized as important.

Bakhtin (Wertsch, 1993) regarded language as a primary mediator and developed several influential concepts. "Utterances" he saw as concrete instances of talk "attached" to everyday use, and he treated these as his unit of analysis. He saw utterances relationally, based on the relationship between a speaker's point of view and the social situation in which the utterance occurred. In the analyses in this dissertation, utterances are seen as being a part of what constitutes the speaker's activity. Bakhtin's notion of "voice," as a process tied to the speaker and their audience that emerges out of dialogues with others, is significant for this dissertation. The voices of the designer and developer are seen as important and the analysis attempts to bring out the voices of these specialists as interacting individuals engaged in a multidisciplinary creative activity. Bakhtin's notion of "dialogicity" addresses relations in speech, implying that one speaker's utterances can be understood only in relation to another's. Bakhtin's approach has directed the analyses of talk in this dissertation toward being concrete, process-oriented, and contextual.

Some researchers have been more explicit about examining the role of artefact use in communicative activities. Wells (2002) saw artefact-mediated action, such as

workplace tools and semiotically-mediated interaction, as discourse that occurs simultaneously and alternately. His research emphasized that each subject brings tools, skills (internalized tools), and artefacts to the discourse and that the outcome of a discourse is both shared understanding, through the creation of newly articulated semiotic artefacts such as accounts, explanations and theories, and new material artefacts. Multidimensional communicative activities develop because of both talk and interaction with available materials/artefacts. He showed that actions with artefacts could serve a different purpose than actions with words and described how, in one instance, explorative artefact use showed *how* to build something. Wells' emphasis on the complementary aspects of talk and artefact-use, the situated nature of communication, and the value placed on artefact-mediated actions is also an important aspect of this research.

Bødker and Andersen (2005), emphasizing the role of artefact use in activity, combined the notion of mediated action from cultural-historical psychology and the interpretation of signs from Piercean semiotics. They concluded that real-life mediation “is heterogeneous (comprising not only traditional controls and displays, pictures and texts, but also conversations and entire activities); dynamic (mediators change through time); and consists of webs of mediators...” (p.356). Viewing conversation as an activity in its own right, Bødker and Andersen saw the object of conversation as the topic of the conversation. However, they also reported that when conversations are mediated by artefacts as well as talk, the topic is both what is said and what is represented by the artefact and, conversely, what is interpreted is both the talk and the artefact. Further, taking a process-oriented perspective they suggested that conversations typically proceed with cycles of topic transformations and interpretations. Bødker and Andersen's mediation analysis is valuable to this research because it acknowledges the significant

role of artefacts in conversation and suggests that there are patterns to be found in analyzing talk- and artefact-mediated activities.

Roth's work (2005) showed how talk and artefact use "makes things happen." When analyzing videos he incorporated "zooming" and "focusing" to attend to both individual and group activity simultaneously as they are enacted through talk and artefact use. His work "get[s] at the ways in which humans produce/reproduce ordinary everyday society" and showed that "not only does social life emerge from such ordinary events, but also its characteristic structures, hierarchical relationships, and even who we are with respect to others, that is our identities are produced/reproduced continuously in seemingly mundane everyday conversations" (Roth, 2005, p. 383). Roth's work emphasized the concrete and dialectical nature of multimediated discourse by holding individual activity, group activity, and society in dialectical tension with one another (meaning all three were considered in the analysis). In this way, Roth has influenced this research.

Artefact Use in Software-Creation Work

Cultural-historical psychology is not unique in its focus on artefacts or its interest in concrete activity. This section surveys a number of studies that have looked at artefacts in order to understand interaction designer or software developer behaviour. The studies reviewed here emphasize the diversity of approaches to researching this topic, the analytical tools used by other researchers, and the results of these reviewed studies.

Suchman (1987), interested in the lack of success many adults experienced when using a certain computerized printer/photocopier with a modestly complex user interface, used an ethnomethodological perspective to understand why breakdowns occur when people use computerized printer/photocopiers. She viewed artefacts as mediators, favoured research based on observable phenomena, and understood actions in the

occasions of their production and use (Suchman, 2000). Suchman's work focused on the local interactional work that produced intelligibility in the three pairs of adults using printer/photocopiers. Suchman's premise was that interpreting the significance of an action is "a collaborative achievement," meaning that she understood the significance of human actions in conjunction with photocopier responses (Suchman, 1987). Her unit of analysis was the moment-to-moment interactions between the pair of adults and the machine and the relation of adult-machine interactions to the adult's interpretation of them. She wanted to know how people take their understanding of the world and reflect it outwardly through their actions. The talk between the adult pairs was recorded along with photocopier responses, converted to table format, and analysed using interaction analysis, an analytical technique I also applied in my research.

Suchman reported that when people interact with computers, they make assumptions about their interactions based on their experience with human-to-human conversations. However, these assumptions led to many difficulties or conversational breakdowns because computers do not share the cultural practices of conversations assumed by humans. Suchman revealed both the assumptions of users and also those of the designers of the photocopier.

When addressing the assumptions of designers, Suchman developed a new understanding of the relation between plans and behaviour. As Suchman argued, difficulties arise in the course of concrete material situations unfolding, as concrete actions and interpretations of the actions interact with the plans that acting people have made. She hypothesized that popular, but false, understandings about the nature of interactions from cognitive psychology had been absorbed by the designers of the photocopier and were implicit in its poor design. Essentially, the problem was that the

machine could only “perceive” a small subset of the users’ actions that triggered changes in state, and missed actions, such as photocopier doors being opened or closed or certain buttons being pressed. Therefore, Suchman concluded the notion of reciprocal action or influence could not be fully achieved in the interaction between the human and machine (Susi & Ziemke, 2001). Additionally, behaviour was better explained by objectives emerging from situations, rather than by preformed plans.

Suchman’s results supported the idea put forward here that cognition can be studied through artefact use in concrete, social circumstances.

What traditional sciences take to be cognitive phenomena have an essential relationship to a publicly available, collaboratively organized world of artefacts and actions, and secondly, that the significance of artefacts and actions, and the methods by which their significance is conveyed, have an essential relationship to their particular, concrete circumstances. (Suchman, 1987, p. 50)

Or, as she stated, “Every course of action depends in essential ways upon its material and social circumstances” (Suchman, 1987, p. 50).

Suchman examined representations of work in software development and analyzed these in relation to work practices and the politics of organizations (Suchman, 1995). Suchman reported that good representations of work can only be determined in a particular situation and relative to other representations. She claimed that representations make things visible so that they can be seen, talked about, and potentially manipulated; the need to do this arises out of the work. Suchman claimed that,

Once we recognize that representations are artefacts constructed from particular social locations and within specific forms of practice, we can expand our concern with the adequacy of representational forms to include ongoing dialogue and debate regarding the various places of representations in work and system design. (Suchman, 1995, p. 63)

Spinuzzi (2003) looked at program code used by developers and used the “interpretation of genres” as a unit of analysis. Using genre ecologies, based on genre

theory, activity theory, and distributed cognition, as a research orientation, Spinuzzi studied software development as a mediated activity, viewing artefacts as belonging to genres that have been adapted or developed by a community to mediate certain activities (Zachry, 2001). Genres are developed within a community and provide a community-centered interpretive view, meaning that communities can be understood, to some degree, through their artefacts. This framework supports thinking about mediation as taking place within a community with its own cultural-historical milieu. It also supports looking at how people reflexively and developmentally use resources to do their tasks.

Genre ecology attends to how official and unofficial genres are adapted, negotiated, and discarded. Spinuzzi, for instance, observed that developers in one part of a company interpreted the purpose of comments in program code differently than workers in other parts of the company (two groups disregarded comments and one did not), resulting in the code being used differently in each of the communities (Spinuzzi, 2003). His work supports the idea that types of artefacts, as genres, have been historically adapted or developed by a community, where communities inhabited, adapted, and extended genres of artefacts. Genre ecology supports an examination of how people reflexively and developmentally use resources engaged in activities by focusing on how artefacts are interpreted. Genre ecology models artefact relationships as an interconnected web (Spinuzzi, 2003, p. 65) represented graphically. By developing genre maps, Spinuzzi also found that some artefacts are central artefacts, used concurrently with many other artefacts. In the developers' case, the central artefact was the program code, which was true for all of the three groups he studied. Spinuzzi also showed that networks of artefacts varied from one development group to another within the same company, reflecting different practices in the different groups.

Robertson's research provides another perspective on cognition and reveals more about the bodily responses and the use of artefacts during design work. Robertson (1997) asked, "How is cooperative design sustained?" To answer this question, she used the phenomenology of Merleau-Ponty, which assumes that the lived experience of the embodied subject is the basis of both experience in the world and agency. Robertson focused on physical bodily responses of designers as her unit of analysis.

Using observation, video-taping and interviews, Robertson conducted a 12-month longitudinal study of eight designers from the same design organization. Her goal was to find out how people created and maintained the cooperative design process. She concluded that "the cooperative design of the product was enabled and achieved by the work the designers did communicating with each other" (Robertson, 1997, p. 341). She argued that physical objects were crucial in the process because they enabled the inclusion of the subject matter depicted or the meaning of the representation in the conversation. She saw work as unfolding through time and across space and achieved by the embodied actions of the designers. Robertson (2000) regarded cognition as embodied interaction.

Robertson's study showed the following taxonomy of embodied actions, which she sees as both group activities and individual actions in relation to: (a) physical objects, (b) other bodies (meaning people), and (c) the physical workspace. In this taxonomy, the items related to artefact use were groups looking at the same thing at the same time, groups organizing shared communication resources, groups creating a shared representation, groups sharing a physical object, groups focusing attention on an object, individuals moving objects, individuals producing representations, individuals highlighting a part of an object, using an object, and pointing to an object.

Robertson (2000) showed how the interactions of designers can be studied in a very concrete, yet uniquely philosophical, manner. Robertson also provided a taxonomy that described the designers' and the group of designers' artefact-oriented actions. Although Robertson provided no numbers, it is easily inferred from her descriptions of the designers' work that there are many of these actions. And, although she does not explain the purpose of such actions, her analysis shows that design work relies heavily on many different types of artefacts and substantial work is involved in creating and managing the artefacts designers use.

Research Questions

Research questions were used to break down and then achieve the main aim of the dissertation—to understand the interactions of designers and developers engaged in the early stages of software-creation work and the role of artefacts in these interactions.

The first question asked was: What can be understood about designer-developer interactions from a historical perspective? This research studies the history of the disciplines of software development and software design focusing on the development of the roles of the developer and designer over time. It identifies unresolved tensions in the roles of the designer and developer and unresolved tensions between these roles.

To understand the role that designer-developer, artefact-mediated interactions play in the workplace, the second question asked: How is the work of designers and developers structured within organizations and what role do artefacts play here? Workplace observations and an analysis of the work observations exposed this structure.

To understand artefact-mediated interaction from the perspective of both the developer and the designer, the third question asked: What meaning do artefact-mediated interactions have for designers and developers? In-depth interviews and an analysis of

interview transcripts revealed the meaning that artefacts have for designers and developers.

A final aim was to understand the interactions between developers and designers that are mediated by artefacts and to identify important features of these interactions in workplace meetings. Here the question was: How do artefacts support designer-developer interactions in meetings? The primary means of answering this question was through an analysis of videos and photos of the artefacts that mediate designer-developer interactions in meetings.

Throughout the research, qualitative methods were used. Qualitative research consists of “a set of interpretive, material practices that make the world visible” (Denzin & Lincoln, 2005, p. 3). The multiple methods used in this dissertation have aimed to “secure an in-depth understanding of the phenomenon in question” (Denzin & Lincoln, 2005, p. 5). Qualitative methods were a natural choice for this investigation because they excel at supporting the direct study of dynamic social contexts (i.e., software-creation work), and representing these through the eyes of the research’s participants (i.e., the designers and developers). Qualitative methods are also excellent for investigations of complex perspectives of individuals, such as the designers and developers’ view of their work.

Regardless of the particular qualitative method used or the level of inquiry, this research relies heavily on case studies and detailed descriptions to develop a deep understanding of phenomena. However, because this research was aimed at making claims of a more general nature, multiple, rather than singular, case studies were conducted in every part of this dissertation.

Methods

Analytical Methods Used

Cole (1997) suggests there is a “need to study culturally-mediated behaviour developmentally” (i.e., over time). He also suggests the need “to conduct research at several developmental/historical levels to analyze the ways in which they intertwine and fuse in human life over time.” As such, this dissertation has been organized into four perspectives, each one both intertwined and separate from the others, but all shedding light on the research topic.

Each perspective is designed around a single research question, but some perspectives are studied using multiple research methods. In all, five methods were selected from an extensive review of methods conducted for the prospectus defence. Methods were selected based on the method’s ability to increase understanding, to analyze talk and artefacts simultaneously, to deal with integrated data, and to deal with large amounts of data. It was also important that research methods were aligned with the purpose of the research, the theoretical paradigm, i.e., cultural-historical psychology, and the situation within which the research was carried out (Terre Blanche & Durrheim, 1999).

This research was conducted from a constructivist-interpretivist paradigm. Relevant underlying assumptions include that realities are socially constructed; that extended interactions between researcher and participants uncovers deeper meaning and insight; that my biases as a researcher should be exposed; that participant’s voices are of value; and that meaning can be uncovered through an indepth analysis of multiple, qualitative data sources (Ponterotto, 2005). Relevant criteria for the evaluation of such research is discussed by Morrow (2005) and will be addressed in the conclusions chapter

of this dissertation. These include: fairness (i.e., honouring participant viewpoints equitably), ontological authenticity (i.e., the effective expansion and elaboration of participant viewpoints), meaning grounded in a contextualized understanding of participant's lived experiences, consequential validity (in this context, the extent to which understanding of designer-developer interactions and software creation activity has been improved by the research), adequacy of the data, and adequacy of the interpretation. The report has been structured following guidelines for reporting qualitative research in Ponterotto (2007).

The methods selected are presented below. The perspectives are then described together with the research methods used to address the research questions.

Activity Systems Analysis

Activity systems analysis (Engeström & Engeström, 1986; Engeström, 1999b; Engeström, Miettinen, & Punamèaki-Gitai, 1999; 2000; 2002) expands on Vygotsky's notion of mediated action and Leontiev's work on activity. This method is frequently used for the analysis of field notes. In activity systems analysis, the unit of analysis is a historically evolving collective activity system. "A historically evolving collective activity system, seen in its network relations to other activity systems [is] taken as the prime unit of analysis against which script strings of goal-directed actions and automatic operations are interpreted" (Engeström, 2000, p. 960). In this dissertation, both individual activity systems and networks of activity systems are the prime unit of analysis within which interacting individuals, rather than acting individuals, are interpreted.

Analyzing work using activity systems analysis identifies the real order in work. Here it was used to cut across an organization's professed organizational structure to expose a network of interrelated, motivated, structured, and purposeful activities. Some

of these were re/produced by the actions and interactions of designers and developers and others had a bearing on designer-developer actions and interactions.

Activity systems analysis identifies key interrelated aspects of an activity while also supporting a holistic view of it. It reveals how teams, groups or individuals in an organization have structured their work through the use and creation of roles, rules and artefacts. The assumption is that activity systems analysis reveals an inherent way that humans organize themselves. The interrelated aspects of an activity are subject, tools, object/objectives, rules, community, “division of labour”, motive, and outcome (see Figure 1). *Tools* mediate between the *subject*, such as a designer or developer, and an *object*, such as a representation of a software product. *Rules*, such as how meetings are run, mediate between the subject and community, namely, the product team. *Division of labour* addresses the creation of various roles within the team including the designer’s and the developer’s. It mediates between the community and its object and determines the focus of various members of a community on different aspects of an object so that collectively they can transform it. This is the sense of the meaning of the word *role* in this dissertation. Other important concepts are *outcome*, which is the result or output of the activity, in this case including a developed software product, and *motive*, which is understood here as the group’s motive or the reason for the activity.

Whatever the element being considered, interrelatedness, rather than the distinctiveness of these elements, is emphasized because each is reflected in the others (forming a system), and all the parts change and shift if one element is altered. For instance, the introduction of a new role or motive could change all other aspects.

Figure 1. Engeström's activity system triangle.

An important property of activity systems is that they are internally contradictory or laden with tensions (Engeström, 2000, p. 960). The notion of contradictions means that an activity systems analysis is explanatory, as well as descriptive, because the resolution of contradictions explains change. For instance, when creating software in multidisciplinary groups, exposing one's work products allows others with differing concerns and skills to influence the work products, if the particular artefact is problematic to them in some regard. From an analytical perspective, identifying contradictions is also valuable because these highlight points of potential growth. The resolution of contradictions can lead to learning or growth within the activity system.

Activity systems analysis can also expose the points of contact between related activities.

Contradiction Analysis

Disturbances are problems, ruptures, breakdowns or clashes in an activity. They are commonplace and those engaged in an activity are generally aware of them. But disturbances occur for a reason and this is what a contradiction analysis seeks to expose. In activity systems analysis collections of similar disturbances are seen as symptomatic of

an underlying tension in an activity system, which commonly occurs because two elements of the activity system do not fit well together (e.g., a mismatch between a person's role and their tools) or there is tension within an element (e.g., two views on the outcomes for the activity).

Persistent systemic tensions are called contradictions. They exist in all activities. Activity theorists find them interesting because they are the sources of change and development in an activity and can therefore explain these elements. For instance, a contradiction might explain the reason for a change in tools or why no progress is being made toward a certain outcome. Contradictions can be minor or more significant. They can influence the entire activity, pulling it in different directions. In extreme forms, they are experienced as a double-bind or unresolvable dilemma (Engeström, 1999b; 2001; Kerosuo, 2001; Kerosuo & Engeström, 2003). A contradiction analysis seeks to identify important contradictions from repeated patterns of disturbances. Put simply, disturbances are like problems, and a contradiction is the explanation for a set of related problems.

Though commonplace, contradictions can be difficult to find because those engaged in an activity are not necessarily aware of them. Engeström suggests the identification of contradictions requires familiarity with an activity system, which leads to awareness of disturbances and then to identification of contradictions, although the latter, he says, is essentially an art (Engeström, 1995). Turner and Turner (2001) suggest the way to identify contradictions is to ask: What forces or requirements have shaped an activity in a certain way? Contradictions can be found in concrete activities such as software-creation work, but also in diffuse activities such as disciplinary activities. Turner and Turner (2001) provide an example of a contradiction in the diffuse activity of all government benefit agencies, which is “the desire that all claimants have unrestricted

access to their entitlements and the need to deter fraud by, in practice, restricting access to benefits” (p. 2). Such pervasive contradictions often appear in an activity as disturbances in the work. For example, government clerks likely experience Turner and Turner’s contradiction on a daily basis because they must be suspicious of every claimant before they may issue benefits. Once a contradiction is found, it is possible to look for disturbances it may be causing in other elements of the activity.

I found that it was easy to find descriptions of disturbances in the literature, but also that it was possible to find some descriptions of contradictions in relevant papers written by cultural-historical psychologists, historians or social scientists. Searching for contradictions already found and described by others could therefore assist the process of finding contradictions in any activity. However, to confirm that contradictions from the literature were relevant, I matched disturbances and contradictions with each other. To identify contradictions from disturbances I asked: Why does this pattern of disturbances occur? and What are the underlying tensions? And to associate disturbances with contradictions I asked: Is this disturbance evidence of the underlying tensions in this contradiction? Finding these contradictions is important because they are systemic and persistent, and they have the potential to explain change or to identify points of possible change in a system.

Interaction Analysis

Although developed as a video analysis technique, in this research I used the principles underlying interaction analysis (Jordan & Henderson, 1995) to analyze both meetings captured on video and detailed field notes annotated with photographs of artefacts (Brown, Lindgaard, & Biddle, 2007; Brown, Biddle, & Lindgaard, 2008b). Interaction analysis supports a more detailed analysis of processes in comparison to

activity systems analysis. It also provides an excellent overview and a more structured analysis than standard ethnographic techniques that involve making direct interpretations from the data. At the same time, it is less detailed than other methods, such as conversation analysis, which are best-suited for analyzing small segments of interactions.

Interaction analysis “emerged over the past 20 years as a distinct form of analysis” that “has been extended to a wide variety of organizational settings” (Ruhleder & Jordan, 2001, p. 117). Interaction analysis helps researchers focus on the mundane and explores “the underlying, usually invisible, fabric of interaction” (Tatar, 1989, p. 108). It is a grounded technique, keeping close to the data. It is also a framework, rather than a prescriptive method. During analysis the emphasis is on the entire interaction within context, and not interaction from the perspective of one of the “interactees” (Jordan & Henderson, 1995). It is nondeductive, meaning that observations are conducted with an open mind and not based on a priori assumptions (Suchman, 1995). It can be used to study concrete, moment-to-moment interactions, to make links between the personal and the social, and to address the role of work representations or artefacts in use (Suchman, 1995). Interaction analysis also “investigates human activities, such as talk, nonverbal interaction, and the use of artefacts and technologies, identifying routine practices and problems and the resources for their solution” (Jordan & Henderson, 1995, p. 39). The technique identifies the beginning and ending of events and how artefacts may play a role here, how events are structured and the role of artefacts in this regard, how events are periodic, how turn-taking occurs, etc.

The roots of interaction analysis are primarily found in ethnomethodology, which is the study of people’s methods, particularly phenomenology and the situated traditions. Other influences have been ethnography (i.e., the qualitative description of human social

phenomena based on field work), sociolinguistics (i.e., the study of the effects of any and all aspects of society on the way language is used), kinesics (i.e., the interpretation of body language), proxemics (i.e., the study of the relation between physical space and social space), and ethology (i.e., the study of innate animal, including human animal behaviour) (Ruhleder & Jordan, 2001, p. 117). Hence, a considerable amount of expertise is embedded in the method. The value in the technique is that it produces accounts of social order and its maintenance based on ongoing changes.

In practice, interaction analysis requires repeated viewings of video recorded material and in this dissertation, on occasion, repeated rereading of notes and the careful examination of embedded photos. An understanding of work practice is then founded in an emergent understanding developed through iterative examination of the data, which is usually conducted with multiple researchers simultaneously. Interaction analysis looks for orderliness and patterns in people's routine interactions. However, the focus is on what is visible or audible on video recorded materials, which are referred to as "collectables." The output of an interaction analysis can be a rich description, tabular data, a theory, or any combination of these.

Several examples of research using this method illustrate the method's value. Suchman (1987) investigated pairs of computer users using challenging printer/photocopiers, as previously described. Her research showed that actions, particularly repair actions, are both situated and planned. Ruhleder and Jordan (2001) used interaction analysis to analyze how delays generate trouble in distributed interactions. They showed that because each party experienced a different conversation in delayed and distributed interactions, it was difficult to build shared meaning. Their detailed analysis revealed exactly why trouble arose. In another study, Brereton (2001)

used interaction analysis in a learning context to study how people make sense of objects and how objects support thinking. A table showing a column of utterances relating to a learner's experience and a column of utterances relating to the learner's developing conceptual model highlighted the dialectic relationship between the abstract (the developing conceptual model of the learner) and the concrete (the learners' experiences with objects). Brereton's work produced a model of learning and identified nine roles that tangible media play in learning, such as "hardware as adversary" and "hardware as prompt." Studies using interactional analysis can therefore produce rich descriptive data, conversations modeled in a tabular manner, and theory.

In the current research, interaction analysis was used to facilitate the analysis of artefact-mediated interactions in meetings involving designers and developers. Although interaction analysis is not a method developed by a cultural-historical psychologist, perspectives that are often associated with cultural-historical psychology, such as ethnomethodology and the situated traditions, inform it. It explicitly includes artefacts within its framework of analysis and it is compatible with a cultural-historical perspective in that it is process-oriented, action-oriented, and concrete.

Interaction analysis, which was conducted multiple times for various parts of this research, differed in three important ways from its recommended application. The technique was applied by one researcher, rather than a group of researchers. The technique was used to analyze detailed, photo-augmented research notes, in addition to videos. The unit of analysis was specifically adapted for cultural-historical psychology and ranged from *a meeting* to *a phase within a meeting* to *an interaction*, whereas normally, the unit is *an event*, which is a suitably vague unit for a multidisciplinary

analysis. When used with larger units of analysis this analysis was referred to as a macro-analysis, and with smaller units a micro-analysis.

Grounded Theory

Grounded theory is a method for developing theories on the basis of text typically derived from semi-structured interviews (Glaser, 1965; Glaser & Strauss, 1967; Strauss, 1993; Strauss & Corbin, 1998a; 1998b; 1998c; 1998d). It is particularly good at exposing what something means from the perspective of an individual or a group of similar individuals. The aims of the analyst using grounded theory are to provide a rationale for theory derived from empirical evidence. Grounded theory serves as a guide for conducting a systematic and careful qualitative analysis conducive to producing theory (Glaser & Strauss, 1967). At the heart of this method is the *constant comparative method of incidents*, which is a comparison of “complete verbal expressions of an attitude or complete acts by an individual or group” in the data (Becker & Geer, 1960, p. 245). Incidents are compared with one another and then coded into categories using either theoretical constructs or in vivo codes (i.e., words or phrases used by participants).

The researcher using grounded theory follows various steps. Initially, through the process of coding, the number of concepts expands and the researcher explores the various concepts for their potential value in terms of their descriptive or predictive potential and relevance to the research question. In the end, a set of concepts and possibly some higher-level categories, their properties, and some initial relations may emerge. This step is known as open coding. In the next stage, in conjunction with continual checks with the data, wherever possible, the categories and their properties are integrated to reveal the relations between the categories. This stage, known as axial coding, marks the beginning of the development of a theory. In the final stage of the method, the theory

is further clarified, simplified and reduced. Reduction allows categories to be combined and given a more abstract construct- or category-level label. Categories are also selected or eliminated at this point depending on their value to the emerging theory. The analysis is completed when the categories are theoretically saturated, which occurs when new information does not alter the emergent model. The technique is both inductive and deductive in that it involves questioning, comparing, as well as hypothesizing and developing categories, properties, and dimensions (i.e., ranges of variation for the properties) (1998a; Strauss & Corbin, 1998d).

Grounded theory has not generally been used in cultural-historical psychology although thematic coding (a simpler coding technique) is not uncommon (Kaptelinin & Cole, 1997). In this dissertation Strauss and Corbin's approach to grounded theory was applied because Strauss (1993) is explicit about how his method relies on his theory of action and interactions in *social worlds*. Also, pragmatism has had a major influence on Strauss's theory of action and pragmatism has close ties with cultural-historical psychology (Cole, 1997). In this study, Strauss's notion of *action* was interpreted in a cultural-historical perspective; *social worlds* were understood as *activity systems*; and *interactions* were understood as *mediated actions between two or more individuals*. As a result, coding was influenced by the assumptions and concepts of cultural-historical psychology.

In this dissertation, grounded theory was used primarily to analyze interview data. However, it was also used for analyzing observational data as in Fitzpatrick's (1998) work, except that in this analysis the observational data consisted of photos embedded in detailed field notes.

Case Studies

The final analytical technique used was case studies (Stake, 2005). This was used as a method for coping with the large amounts of data and was used in conjunction with the previous three methods. Wherever it was used, the most complex, difficult, lengthy, or interesting case was analyzed in detail, followed by more brief analyses of other cases that had the potential to contribute or modify the results obtained from the larger case study.

Participants in General

The four parts of this research were designed to facilitate triangulation of the results (1999b; Seale, 1999a), where triangulation is understood in qualitative, constructivist-interpretivist, rather than quantitative post-positivist terms. Hence each part of this dissertation provides a greater understanding of designer-developer interactions through artefact use and therefore a new perspective. No participants were required for part one, the historical review. The other three parts of the dissertation were conducted in parallel in each of the organizations. In total I recruited eight organizations. In each organization I required a designer-developer pair and a project at the concept-design stage.

Sampling for organizations, participants and projects was carried out opportunistically (Luborsky & Rubinstein, 1995). Many barriers prevented participation by organizations and introduced bias into the research. Twenty-eight organizations in Ottawa, Toronto, and Montreal were approached: (a) six did not qualify because their projects were inappropriate (no development, no design), not novel, or they lacked a designer; (b) three were eliminated because security issues could not be resolved; (c) one site was eliminated because the developer worked remotely. (d) In 10 organizations the

lack of support from one or more key individuals prevented the research from proceeding. Eight organizations participated, and in these eight I had at least one personal contact, usually with the designer, although once with senior management. Contacts came through two organizations: a) the Capital Computer-Human Interaction Group (CapCHI), a local organization of professional designers, which is affiliated with the Special Interest Group in Computer-Human Interaction (SIGCHI), of the Association of Computing Machinery (ACM). I have supported CapCHI for many years (including serving on the executive) and b) the Human-Oriented Technology lab (HOTLab) at Carleton University where I am based. To enter the organization I required the consent of the designer, the developer, their immediate managers and, if relevant, their managers' manager. With these approvals, I entered the workplace, where as it turned out, everyone was helpful and I had full access to all employees and artefacts.

In each organization, data were collected from one designer, one developer and, wherever possible, one of their managers. The designer and developer were engaged on the same project and interacted with each other regularly. Anyone who self-identified as a designer or developer was eligible to participate, as no standards exist for classifying a person as a designer or a developer. The participants for all parts of this research were the same pairs of designers and developers.

Apparatus

I used an Olympus digital voice recorder DS-30 to record stereo-quality interviews; a Panasonic DVD video camera model no.VDR-D310 with stereo-quality voice recording to record meetings, and a Canon PowerShot A85 digital camera to take photos. Canon's standard photo album software was used to initially arrange and annotate photos.

ATLAS.ti.5.6, a qualitative analysis tool, was used to code and query meeting notes, interview transcripts, images, and video meeting transcripts. Prior to entering my observation notes, I inserted annotated photos at the exact point in my notes where I took each photograph. This meant I had immediate access to the context in which each artefact was used. I also entered transcribed video data with photos captured from the videos. ATLAS.ti allowed me to manage this nonhomogenous set of data.

Demographic information was entered into a spreadsheet to explore relevant attributes, their ranges, and any relationships with the findings of the studies. Microsoft Excel was also used regularly for creating big-picture perspectives of ATLAS.ti data and to amalgamate memos of data relating to individuals' interviews and meetings.

To communicate with participants formally, I used information sheets, consent forms, demographic questionnaires, debriefing forms, and a thank-you letter. The entire study was approved by the Carleton University Ethics Committee for Psychological Research.

When seeking formal consent for my research, I described my project to participants and provided them with a managerial consent form that their managers could sign to allow me to enter the organization. In three organizations I signed nondisclosure agreements.

At the beginning of the research, participants signed an overall consent form and filled out a demographic questionnaire (see Appendix A). Before I proceeded with each of the four aspects of the research, the participants signed shorter consent forms and read an information sheet that outlined the process for that part of the research and their rights. After the data were collected, they signed a general debriefing form and received a thank-you letter.

Procedure in General

Although I was friendly during my observation sessions, I tried to remain in the position of an observer to not disturb the work of my participants. Throughout I tried to remain nonjudgmental as my limited sampling at each site, the diversity of the organizations and products, and my lack of access to the completed software products precluded any definitive comparative statements about the quality of any organization's project work. Data collection was conducted on site at each of the eight participating organizations. For all the interviews I recorded the audio and photographed artefacts shown to me. I found all of my participants, save three, to be very open. The three participants who were not quite so open were less experienced and concerned about what they could and could not tell me based on their organization's rapidly changing security policies. At four of the eight organizations I also observed, took notes, and video recorded meetings. At these four organizations no barriers were placed in the way of my research, allowing the videotaping of all relevant meetings that took place, save one instance where I was asked not to attend a meeting because of sensitivity stemming from the presence of three levels of management. Other than that, my access to data was restricted in only minor ways, usually because of concerns regarding client confidentiality that were easily resolved. For instance, I double-checked with participants before taking a photo of a document marked confidential.

Although I tried to collect data only until necessary, large amounts of data were collected. To manage the volume of data I selectively analyzed the most complex or most typical data, and analyzed only to the point of saturation. I also analyzed in larger chunks than I might have done with lesser amounts of data. Due to the abundance of data, the

evidence is also reported selectively in this dissertation, avoiding exhaustive descriptions of all the evidence.

To the extent possible, data collection and analysis overlapped. For the analysis using grounded theory, sampling terminated when saturation occurred. For the activity systems analysis, sampling was considered sufficient when a full description of the activity system was obtained, including the identification of significant contradictions. For the interaction analyses, sampling terminated when no new information relevant to this dissertation was found.

To study designers and developers interacting, multiple perspectives were taken, each one focused on a different question and consequently a different type of data. The four questions each take a different perspective, as is suggested by a cultural-historical study of the mediated interactions of individuals. These perspectives are: historical, cultural, participant-centric, and artefact-centric. One chapter has been written analyzing each of the data types. The four data types are (a) literature on designing and developing, (b) field notes and photos, (c) interview data, and (d) meeting videos. Each type of data was analyzed one or more times. The analyses were conducted at a variety of levels and are independent and complementary. At each level, a different process or *history* was examined.

The unit of analysis in this dissertation changed with the process being studied. The unit of analysis provides the focus and in this dissertation all the units of data analysis included the designer, the developer, and talk- and artefact-mediated interactions. The units, at each level, were therefore carefully chosen to capture all the elements relevant to this dissertation. For example, when studying at the level of activity, I used meetings as my unit of analysis because in meetings designers and developers talk

about creating software using artefacts. Using meetings as a unit of analysis means I filtered out all other activity I observed, such as deskwork, to focus on one element—the development of the activity from one meeting to the next. At each level, the unit I selected identifies the focus of the analysis with the element that was observed changing over time.

Perspective 1: Designer-Developer Interactions Understood Historically

The primary research question of relevance here was: What can be understood about designer-developer interactions from a historical perspective? To answer this question a historical review and contradiction analysis of the disciplines of the designers and developers was conducted. The emphasis in this historical review was on the formation of the role of designers and developers, which includes their relationship to one another and the artefacts they have used in their joint work. The notion of contradiction from Engeström's work was used to highlight tensions or contradictions within and between the roles of designers and developers. Table 1 details the process examined, the unit or focus, and the data and analytic methods used for this part of the dissertation.

Table 1 *Perspective 1-Designer-Developer Interactions Understood Historically*

Process examined	Unit examined over time		Data and analytic method
	General	Specific	
Defining the role of designers and developers	The role	The role of the designer and the role of the developer	Papers on the history of software design and software development were analyzed using contradiction analysis

Perspective 2: Designer-Developer Interactions and Activity

The driving question behind Perspective 2 was: How is the work of designers and developers structured within organizations and what role do artefacts play here?

Procedure for Observing Software Work

Observational studies are well-planned events (Adler & Adler, 1994; Banister, 1994). The purpose here was to reveal activities sustained through designer-developer interactions. However, observing in a way that would produce valuable data to reveal the activities in the workplace required careful planning. The challenges in observing software-creation work were that software-creation activities are complex, that they involve many people in various roles working on multiple projects at once, and that they are directed at an object with unique and difficult inherent attributes, such as invisibility and intangibility. Further, they involve many abstractions, they involve a multiplicity of motivators, and they are not easily observed because much of the action can happen quickly or with remote participants. As a result, I controlled some aspects of my observational work by focusing on one project only, by excluding cases where designers and developers routinely interacted remotely, by focusing on only the initial phases of the software-creation activity, and by considering only business motivations for the activity.

To collect observational data I shadowed one designer and then one developer in the organization within which they worked. I repeated this process at four organizations. Participants were shadowed for one to three days, and during this time I was able to collect sufficient data to understand their work context. By shadowing one designer and then one developer on the same project, I collected two views of the same project work. In each organization I shadowed the designer first because this made it easier for me to get an end user's perspective on the project.

I began my observations by asking my participants to articulate their conscious goal as they worked. Then, positioned behind my participants, but with a good view of any artefacts they used, I began observing, photographing, and taking notes. I tried not to disrupt the normal flow of work. However, I did have innumerable participant-initiated chats, as many of the participants were eager to share their views and curious about my research. I watched participants work at their desk and followed participants any time they interacted with other members of the project team and not just when designers and developers interacted with each other. My visits were organized so that I observed as much of the work on a single project as I could, meaning that I entered the workplace only on days when work on the product I was interested in was occurring, and avoided days when my participants were working on other products or projects. I observed my participants in a variety of situations, ranging from working at their desks using and preparing artefacts, to interacting with the other members of the project team in meetings, to desk-side visits with colleagues, and to informally interacting with each other in settings like lunchrooms. My participants and their project teams were all friendly, open, and rapidly adjusted to my observations, photographing, video recording, and note-taking.

The participant's actions were recorded as I observed them. I asked participants to state their goals as they worked and I recorded these too. My notes also captured elements of relevance to the activity (tools in use in the organization, people, roles, rules in the workplace, and so on). I observed on 23 days for over 150 hours. Two hundred and eighty-six pages of notes were taken during the observation period, which is approximately two pages per hour.

Four hundred and forty-five photographs were taken of artefacts. At my first site, I had planned to collect screen shots and printouts of artefacts with the help of my participants, but I quickly realized that this was disturbing to their work relative to taking photographs of screens and the documents they were using. Photos had excellent fidelity; they also captured many artefacts that I had not initially considered and artefacts that were not digital (e.g., sketches on whiteboards) or that were large (e.g., the paper sketch of a game environment that filled an entire boardroom table; the arrangement of piles of papers on the desk and floor space of one designer categorizing report types). I was careful not to photograph people or sensitive information and, on one occasion, deleted a photo of a sketch upon request.

Perspective 2 explored how the creative work of producing software happens materially. This study explored design and development work within a network of organizational-level activity systems. This provided the rationale for designer-developer interactions and identified key interactivity and intraactivity artefacts. Engeström's activity system model was used to reveal the activity systems in the four organizations where I observed.

Summary

The outcome of Perspective 2 was an understanding of how designers and developers organize their joint work and the role artefacts play in interactions between the two specialists whose shared objective is creating a new software product. The nature of product design and development work was revealed and its relation to other work in an organization. Table 2 is a summary of this phase of the research.

Table 2 *Perspective 2–Artefact-Mediated Interactions and Activity*

Process examined	Unit examined over time		Data and analytic method
	General	Specific	
Software-creation work within the context of other work in an organization	An activity	Activities related to software creation	Observation notes and photos from four organizations were analyzed using Engeström's activity systems analysis

Perspective 3: The Meaning of Interactions to Designers and Developers

Perspective 3 was designed to answer the following question: What meaning do artefact-mediated interactions have for designers and developers?

Interviews and Interview Scripts

The interviews with participants were guided through the use of two semistructured interview scripts (see Appendix C and Appendix D). A semistructured interview style (J. A. Smith, 1995) builds a conversational partnership (Rubin & Rubin, 1995) between the participant and the researcher. One script was about the value of artefacts, the other about the software-creation activity in the organization and how it had changed over time. Both interview scripts followed a similar structure. The first questions were easy to answer; subsequent questions had more depth, and the final questions were light. Most questions were open-ended and substantial probing occurred.

The interview on artefacts focused on one specific project, one team, and one set of artefacts, to keep the interview grounded and meaningful. The interview on design

activity changing over time focused on understanding large shifts in design and development work in the organization and the reasons for these shifts.

Procedure for Interviewing Participants

The first interview script was used with the designers and developers. The other, on how software-creation work has changed over time in an organization, was used with managers who were senior designers and developers. I also used a shorter form of this guide with the designers and developers whenever possible. The content overlapped significantly because the first question of the second script asked about the specific project and the artefact-mediated activity that was currently in place that would lead to a finished product. In the end, I used interview transcripts from both interviews to explore the meaning that artefacts had for designers and developers. I interviewed 8 designers, 8 developers, and 1 manager using the first script, and 4 managers, 2 designers, and 3 developers using the second script, for a total of 26 interviews with 20 different people. In four organizations, I conducted two interviews on software design and development work and in the other four organizations I conducted three interviews. One additional interview using the first script was conducted with a designer in one of the organizations on a different product. The longest interview was 2 hours, the shortest 15 minutes. The average length was about 45 minutes.

The 26 interviews were all conducted in the participants' workplaces. The interview was recorded and notes were taken throughout the interview. To begin the interview I clarified ambiguous terms such as "artefact." During the interviews interviewees showed me specific artefacts as they spoke about their work. These were photographed at the time they were shown. At the end of the interview, I asked if the person had a preferred pseudonym. I also asked if they would like to see the results of my

analysis (all of them did) and thanked them for their time. Notes about the interview were written as soon as possible after each interview, sometimes in my car or the company cafeteria. Photos were immediately entered into a photo album and annotated in the evening after I had left the workplace. These I analyzed in conjunction with photos taken when observing participants at work, a process I went through for a separate analysis in Perspective 4. All interviews were recorded and transcribed.

In half of the organizations I was also able to spend approximately 30 hours prior to the interview observing the interviewee at work, a process that was conducted primarily for Perspective 2 of this dissertation. This helped build rapport and it was excellent preparation for understanding the material covered in the interviews. In this analysis I used my observations to confirm that what my participants' reported on how they interacted with others correlated with my observations of their interactions in the workplace.

The interview analysis was conducted at the level of the individual. In other words, the designer or developer's perception of their work was examined, especially in relation to their use of artefacts and joint work. An adapted version of grounded theory was developed for this research, which was influenced by principles of cultural-historical psychology. This was used to identify categories, concepts, mediators, and relations between these as in a previous study (Brown, 2005), and is described fully in the chapter documenting Perspective 3.

Summary

Perspective 3 studied the perception of the software-creation activity as understood by the designers, developers and their managers, who themselves were senior designers and developers. The study is summarized in Table 3.

Table 3 *Perspective 3—The Meaning of Interactions to Designers and Developers*

Process examined	Unit examined over time		Data and analytic method
	General	Specific	
How designers and developers perceive software-creation work	Perceived activity	Perception of the software-creation activity for the project	26 Interviews analyzed using grounded theory

Perspective 4: Designer-Developer Interactions in Meetings

This part of the research, which involved three studies, was designed to answer the following question: How do artefacts support designer-developer interactions in meetings? The meetings of designers and developers at work were captured on video and analyzed.

Procedure for Video Recording Meetings

Video recordings (Bødker, 1996; Harper, 2005) of meetings and photos of artefacts were collected for this part of the study. Four of the eight designer-developer pairs were video recorded in meetings as they worked. When entering the workplace I confirmed that my participants were still comfortable with me video recording their work. I provided instructions to the participant and had participants sign the informed consent form. When meetings were planned in advance, I also sought the informal consent of others attending the meeting, which I always received. Prior to a meeting, I set up my recording equipment and, if necessary, restated my objective and explained why I was at the meeting. Because they were all working on the same project, meeting attendees had discussed my work in advance and had informally agreed to allow me to video record them at work prior to my arrival. I tried to make people comfortable by only capturing the artefacts used in the meeting in my videos and, rather than speaker's faces,

their hands on the table. Videos of speakers' hands helped me to identify them when transcribing. Video sessions varied considerably in length (five minutes to two hours), locale (meetings rooms or desks) and the number of attendees (two to nine). Thirteen meetings were thus captured, generating approximately 11 hours of video material. Most meetings occurred in meeting rooms, but in one organization designers and developers met only at each other's desks.

Study 1: Mediating Artefacts

This study was used to analyze the artefacts used in designer-developer interactions that were captured in videos and in the photos taken during observation sessions that were embedded in field notes or interview transcripts. Images of artefacts used by designers and developers on eight different projects were used in this analysis. Grounded theory was used to code the artefacts and identify categories that were commonly used in designer-developer interactions.

Study 2: The Development of the Activity from Meeting-to-Meeting

The second study looked at the development of the activity from one meeting to the next. Meetings were a key method the team used for progressing design work. This study explored the flow of the software-creation activity from one team meeting to another. Videos and detailed, photo-annotated field notes of workplace activities were analyzed using macro-level interaction analysis (Jordan & Henderson, 1995; Suchman, 1987) to identify patterns of artefact-centred activity between meetings at one organization.

Study 3: Creative Work Within Meetings

Perspective 4 developed an understanding of designer-developer interactions at lower levels: what happens within a meeting, a meeting phase, and with designer-

developer artefact-mediated actions. Each analysis aimed to provide new insight into the work of the designer-developers and their use of artefacts in their interactions.

The videotapes of the designer-developer interactions were analyzed using macro-level interaction analysis (Jordan & Henderson, 1995; Suchman, 1987) using a *meeting phase* or *moment* as a unit of analysis. The phases were then reanalysed using *mediated interaction* and then the *automatic use of an artefact* as the unit of analysis. This produced a descriptive account of the day-to-day work practices that detailed the moment-to-moment, conscious, and automatic actions of the designers. The focus was on what artefact use accomplishes or supports in the interactions.

Summary

Perspective 4 studied artefact use in meetings that involved designers and developers. The studies are summarized in Table 4.

Table 4 *Perspective 4—Meetings Involving Designers and Developers*

Process examined	Unit of analysis		Data and analytic method
	General	Specific	
Study 1: Artefact-mediated actions	Mediating artefacts	Mediating artefacts used in designer-developer interactions	Observation notes, interviews, and photos from eight organizations were analyzed using grounded theory
Study 2: The software-creation activity	A meeting	Meetings involving designers and developers	Videos, observation notes, and photos from one organization's meetings were analyzed using interaction analysis
Study 3: One meeting involving a designer and developer	Phases, actions and operations	Phases, actions and operations in a meeting	A single video recorded meeting from one organization was analyzed in detail using interaction analysis, grounded theory and contradiction analysis.
Meetings of designers and developers	A phase or moment	Phase of a meeting	13 video-recorded meetings from four organizations were analyzed using interaction analysis

Specific Organizations, Participants, and Software Projects

As shown in Table 5, the eight organizations in this research varied across many dimensions including organization type (government to business), size (a large international corporation to a tiny start-up), their clients (a single government department

to businesses world wide), and their preferred design and development process (waterfall to Agile to no process at all). In four of these organizations (Jaba, Talia, Quadro, and Telespace) I was able to observe and video record meetings as well as interview participants. To protect anonymity, pseudonyms have been used for the organizations.

Table 5 *Characteristics of the Participating Organizations*

Organization pseudonym	Approximate number of employees	Market	Domain
Jaba	40	Canadian	Learning games
Talia	15000	Global	Communications
Quadro	1000	Canadian	Service industry
Telespace	50	Canadian	Service industry
Onu	100	Global	Business products
Kadel	3	Canadian	Social media
Zambee	1000	Canadian	Service industry
Edgeworks	1000	Global	Business products

In each organization, I followed the development of a single product. Products varied across many dimensions, including the type of product (business applications to service applications to games to courses), product maturity (brand new to an add-on to a well-developed product), user interface style (Web-based or graphical user interfaces), and end users (ordinary consumers to financial analysts). Product details are listed in Table 6, using pseudonyms for the products. In all cases, I observed the products being developed in the very early, and particularly interactive and messy, phase of the creation of a software product or feature, a phase sometimes referred to as concept design.

Table 6 *Characteristics of the Software Being Created in the Organizations*

Product pseudonym	Product	Client	Users	Process
PlayLearn-it	A game that supports learning	Canadian organization	Canadian employees in various organizations	Collocated, small team (6), highly collaborative, highly iterative
Config-it	A software product for network service configuration	Telcos and ISPs located around the globe	Telco and ISP operators worldwide	First Agile project within the organization. Team was trialing Scrum
Calcul-it	A specialized calculator	Canadian Gov't watchdog organization	Canadian consumers and government employees	Team using many XP practices
Sell-it	A complete redesign of a point of sale terminal	Large Canadian organization	Canadian managers and service personnel at Canadian outlets	Waterfall, with plenty of revisits to previous step
Share-it	A video-sharing application	An Internet service provider	Clients of the Internet service provider	Collocated, small team (3-5), highly collaborative, highly iterative
Teach-it	A online course developed from a face-to-face course	Government department	Government employees	Waterfall process
Reduce-it	A simple feature within an established product	Businesses around the world	Business analysts	Waterfall process for the particular product, but company moving toward Agile

There were 20 participants in total, including 8 designer-developer pairs, and 4 managers of the designer-developer pairs. Because people had multiple roles, I interviewed 12 designers, 8 developers, 2 project managers, 2 managers of designers, 2 managers of developers, 1 chief technology officer, and 2 business representatives, hence I was able to understand software creation activity from many perspectives.

Designer and developer participants varied greatly. As displayed in Table 7, the 11 designers represented a broad range of designer types as reflected in the varying roles they used to describe themselves, such as content designers, learning designer, user interaction designer, user interface designer, interaction design team lead, senior user interaction designer, user experience designer, graphical user interface (GUI) designer and information technology (IT) systems analyst. Because I was interested in changes over time, I asked how long they had been employed in their current organization, how long they had been in their current role in their current organization and any previous organizations, and how long they had been working in their discipline in any role whatsoever. The number of years they had been in their current organization ranged from 4 months to 14 years. The number of years in their current role in any organization ranged from 6 months to 22 years. Data on how long they had worked in their discipline in any type of designer role (i.e., designing interfaces, testing, prototype building, technical writing, analysis, project management, graphic design, etc.) proved to be most relevant. The least experienced designer had 2 years of experience and the most experienced had more than 40 years. Note that Kate, the 12th designer I opportunistically interviewed and whose data I use selectively, is included in Table 7, even though I could not interview the developer she worked with.

Table 7 *Characteristics of the Designers*

Pseudonym	Professed role	Years in organ- ization	Years in role	Years of experience	Education
Alice	Content designer, learning designer, project manager	0.5	0.5	2	MA Psych (HCI), BA Psych
Kate	User interaction designer	2	2	2	MA Psych (HCI)
Jay	User interface designer	0.3	1	6	BA Psych (HCI)
Sean	User interface designer	14	8	8	BA, BID (Industrial Design), Human factors courses
Muriel	Interaction design team lead, Common look and feel expert	6	5	10	BID (Industrial Design)
Owen	Chief Technology Officer	3	3	10.5	PhD, MSc, BSc, BA, diploma, certificates
Nick	Senior user interaction designer	8	11	11	College courses in print production, programming, UI design
Mason	Interaction designer	2.5	10	12	BCOM (IS), M.A. (Psych), Multimedia degree
Frank	User experience designer	1	10	13	Electronics engineering diploma, BID (Human Factors)
Massie	Learning design specialist	3	5	33	MA (Education)
Roberto	GUI designer, strategic marketing concepts	1	22	40	Diploma, certificates in fine arts and cartooning
Alan	IT Systems analyst / consultant / point of sales expert	n/a	n/a	n/a	Not available
Averages		3.8	7.0	13.4	

Note. Years in role means years in current role. Years of experience means years in any designer role.

As is evident in Table 8, the eight developers comprised an equally diverse group. Their titles included game developer, software developer, software designer, lead GUI designer, learning technology consultant, founding developer, software architect and Java developer. The number of years they had been in their current organization ranged from 6 months to 16 years. The number of years in their current role ranged from 3 to 22 years. The total number of years in the discipline of software development, in any role (coding, testing, technical writing, project management, etc.), was 3 years for the least experienced developer to 22 years for the most experienced developer. More details, including averages, can be found in Table 8.

All of the designer-developer pairs worked face-to-face. All were established professionals in real workplaces working on real software products. The research had to deal with the inevitable complexities of this reality because the work through which software products are conceived is highly creative, irregular, unpredictable, and varies significantly from one project to the next.

Table 8 *Characteristics of the Developer*

Pseudonym	Professed role	Years in organi- zation	Years in role	Years in discipline	Education
Jason	Game developer	0.8	3	3	BSc (Comp)
Kylie	Software developer	7	7	7	BSc (Comp, Minor in Business)
Damian	Software designer	7	7	10	BSc (Comp)
James	Software developer	8	10	10	BSc (Comp), training in hardware support, software support
Steve	Designer/ Developer; team lead	1	10	15	MPA (Master in Public Administration) incomplete
Eric	Learning technology consultant	16	16	17	BSc (Comp), Theory of Education certificate
Jack	Founding developer, software architect	0.5	20	20	BSc (Comp)
Wil	Java developer	5	10	22	BSc (Software), some hardware, graphics
Jane	Developers' manager	19	10	19	BSc (Comp)
Averages		7.1	10.3	13.7	

Note. Years in role means years in current role. Years of experience means years in any developer role.

Personal Bias

According to Denzin, “Qualitative research is a situated activity that locates the observer [the researcher] in the world [of their research]” (Denzin & Lincoln, 2005, p. 3). This statement is important because it raises questions about both the biases of the researcher and the authority a researcher has in the domain of research. This section therefore locates me in the world of my research.

I have had 6 years of experience as a computer scientist and software developer working in a research lab and writing telephone switching software. This was followed by experience in academia. I was a senior lecturer in computer science and software engineering at Victoria University of Wellington in New Zealand between 1987 and 2001. During this time I became interested in the field of human-computer interaction (HCI). I have been actively involved in HCI since 1995. In 2001, I decided to focus solely on user interaction issues and became a student in the human-oriented technology lab (HOT Lab) at Carleton University. From 2001 until recently, I was an executive member in two professional organizations involving designers in Ottawa. I have been interested in activity theory since attending a workshop for designers in the 1990s with Victor Kaptelinin and Bonnie Nardi and have had an interest in cultural-historical psychology since reading Cole’s book *Cultural Psychology* (Cole, 1997). Professor Lloyd Strickland, a Bekhterev scholar, guided my entry into the literature of Russian psychologists. Also, a course I participated in on Mediational Theories of the Mind offered online through the University of California, San Diego by Michael Cole in 2006 was a valuable orientation into the literature.

My insider's perspective on both the software and design disciplines is both asset and liability. On the one hand, I have a good deal of domain knowledge which gave me insight into my participants' goals as I observed them at work, and allowed me to understand the challenges faced by both designers and developers. However, because good qualitative research requires fresh eyes and an open mind (Strauss & Corbin, 1998a), throughout my work on this dissertation I have attempted to refrain from judging individuals, projects, and organizations, making comparative statements, or testing hypotheses about any aspect of the work I was observing. Based on the rigorous criteria used to select the organisations, projects, and people for this series of studies, I felt it was safe to assume that what I had the privilege to observe and explore in my research, in its various messy and imperfect manifestations, were highly trained individuals and teams producing successful software applications for their organizations in a manner well-adapted to their situation. This allowed me to focus on the larger question: How did they do this and what does this say about designer and developer minds in action?

Results and Discussion Perspective 1: Mediated Interactions and Historicity

Activities have rich histories that can be profitably explored for insight into current work practices. This phase of the research asked: What can be understood about designer-developer interactions from a historical perspective? To answer this question, a historical analysis was conducted to find contradictions inherent in the roles of designers and developers, and in designer-developer interactions, that would shed some light on modern software creation activities.

Many possible histories are relevant to the practice of software creation. One is the history of the software-creation activity within an organization, and another is the history of how software-creation work has changed generally over time. The history explored here was the formation and development of the disciplines from which the roles of designers and developers emerged, a history which directly addressed the research question.

Disciplines are activities (Engeström, 1999a); they are a diffuse and global dialogue spanning conferences, symposia, workshops, publications, and the events of professional organizations. The result of disciplinary dialogues in disciplines of design and development is to position the work of designing or developing in academic or industrial settings. These dialogues also develop and disseminate knowledge about designing or developing through the publication of reports or studies of the work involved in designing and developing, and reports or studies of the roles of designers or developers. However, these dialogues are not without disturbances, which can reveal contradictions (Engeström, 1999b).

Through an analysis of such contradictions, I obtained a more objective perspective of these disciplines and a better understanding of the roles of designers and

developers, and the issues that arise when they interact. The better understanding was not about what a designer or the developer does, but was about inherent challenges of being a designer or a developer.

The first step in such an analysis normally is to become familiar with the activity being analyzed, which was bypassed in this study because I was already familiar with the disciplines of both designers and developers.

The next step is to find disturbances, which was achieved through a literature search that also revealed some contradictions or underlying explanations for disturbances. Many disturbances were reported in papers written by highly respected, long-standing members of the disciplines of designers and developers who were reflecting on the problems of their disciplines. Historians and social scientists writing about software-creation work also revealed disciplinary disturbances, but in addition, they discussed disciplinary contradictions because of their skills in identifying systemic tensions.

In presenting the results, I first report on disturbances and contradictions regarding the shared object of the designer and the developer, i.e., the software itself.

Then I identify disturbances and contradictions within the developers' and designers' disciplines. After associating the disturbances (problems) and contradictions (explanations), I then specify the impact of the contradictions on the roles of the developer and the designer. My reasoning here is firstly that serious disturbances in the disciplines (e.g., disputes over identity, theoretical underpinnings, and so on) reveal systemic contradictions, i.e., the reasons for a dispute expressed as two truths that are in tension with one another. Secondly, because systemic contradictions are pervasive in an activity they therefore also appear in roles, but in doing so they are transformed. To conclude this part of the analysis, I therefore transform the contradictions I found in the

disciplines of designers and developers to contradictions in the role of the designer or developer.

Finally, to conclude the presentation of results, I report on disturbances that have been reported in software creation work that involve both the designer and the developer and the artefacts that have been proposed for addressing these disturbances. I close with a list of contradictions that relate directly to designer-developer interactions.

In the conclusion of this dissertation, I will show how the contradictions I identify in this historical analysis were reflected in the daily lives of participants.

The Nature of the Software Object

Today, software-creation work is a multidisciplinary activity involving designers in the creation of the user interface and developers in the creation of functional code. However, software development is a new activity for humankind and software is a new object created by this new type of activity. Unlike literature or mechanical engineering, which have centuries of history, the entire history of software creation is less than 65 years old (I consider only software that was executed on a computer and not just written programs). Mahoney, a historian who studies software development, speaking about computers said, “The computer embodies a historically unique relation of thinking and doing. It is the first machine for doing thinking” (Mahoney, 1990, p. 325). But computers “do their thinking” through the software that runs on it. As Mahoney noted, “Between the mathematics that makes the device theoretically possible and the electronics that makes it practically feasible lies the programming that makes it intellectually, economically, and socially useful” (Mahoney, 1988, p. 119). The demand for software has spawned the creation of new areas of expertise, and, therefore, new experts whose goals are to jointly address the task of creating software.

At the organizational level, the processes for strategizing, specifying, managing, selling, and servicing software are not the same as the processes used for traditional products produced through an industrial design process. The state of software development is difficult to track, software can move from one computer to another easily, software does not wear out in the same way as traditional products, and software is maintained differently (e.g., it can be continuously upgraded).

Also, software challenges many established practices and institutions that have been developed for more traditionally designed products. Keet (2004), a pioneer in the industry, recalls that “All through the early years of the software industry, we debated a seemingly simple question: What is software? The answer, once we could agree, would help determine our approach to intellectual property protection, taxation, contracting and public policy issues” (p. 31). Or as Mahoney (2000) said,

There is something different about it [software]. Its science is different from other sciences, its engineering differs from engineering in other technologies, its workers are different from workers in other industries, its economics calls for different models, it poses different challenges to those who want to assure themselves of its dependability, whether as producers or consumers. (p. 285)

Although software-creation is challenging because it is a new kind of work, it is also challenging because of the nature of the software object itself. Mahoney (2000) calls software a “seemingly amorphous object ... which is invisible and intangible, yet produces visible and tangible effects in the world” (p. 285). Because software is amorphous (i.e., we cannot see software on memory devices or touch it because it has no form), it is difficult to ascribe attributes to software as one might to other designed objects such as a teapot or a mattress. Users cannot tell if the underlying program is a tangled mess of “spaghetti-code,” or an elegant, well-thought out arrangement of data and instructions. Yet when software executes or runs, and processes input (e.g., keystrokes,

voice commands, or streaming video) and produces output (e.g., information, animation, or tactile feedback), it is experienced. Unlike the program, the user can ascribe attributes to the interface, such as whether it is appealing or usable, or to the software in general, such as whether it is functional or robust. Thus, software has a dual nature as something whose user interface is experienced when it is run and also as something that is invisible and intangible. On software projects with a significant user interface component, the designer is, in general, concerned with aspects of the experiential nature of software, while developers are, in general, concerned with the invisible and intangible aspects. But, the two responsibilities are inevitably intertwined because the user interface is implemented through programming and the programming logic and back end are experienced through the interface.

DeGrace described the work of creating software as a *wicked problem* (DeGrace & Stahl, 1998), a term coined by design theorist Horst Rittel (1973) to describe the common nature of very difficult problems in widely varying domains such as architectural design and global warming. Wicked problems have challenging requirements, the steps to solving wicked problems are ill-defined, the process of solving wicked problems can reveal more wicked problems, and solutions to wicked problems are difficult to find and evaluate (Rittel & Webber, 1973). Wicked problems impact the entire team of people assembled to solve the problem, and in the case of software, not just the developer or designer.

With software projects, requirements (either needs or constraints) are not always known, can be contradictory, and can change because software systems are part of a complex, changing social environment. During the course of a project changes may occur that impact the project, including the circumstances or opinions of stakeholders or end

users, organizational practices, institutions, or laws. The fact that software reflects so many different aspects of the social environment makes it different from many other designed products. Ambiguity in the requirements impacts all members of the team. For designers this means changes to the interface, and for developers, changes to the code to implement the interface or to the programming logic.

There are no well-defined precise steps to producing a software solution—certainly none have emerged in software-creation work to date, although many general processes for creating the interface, writing the code, producing the requirements, testing the software, and so on, have been proposed.

Surprisingly, new software problems can be discovered in the course of solving a single software problem. Brooks (1987) says software projects “can transform unexpectedly from the familiar into horrors” (p. 10). Mallalieu, Harvey, and Hardy explain why: “Because of the interconnectedness of things, one’s investigation into a wicked problem might reveal deeper underlying causes [of the problem], or simply other factors at the same [social, technical or technological] level, which are embedded in different issues [i.e., new problems are revealed]” (Mallalieu, Harvey, & Hardy, 1999, p. 41). Software-creation work is, therefore, an activity where it may be discovered that a new software project needs to be launched in the course of solving a software problem, unlike products produced through an industrial design process. In designing a car, for example, one does not suddenly discover that another full car must be designed before the first car can be completed. In software one can discover that a part, that is equally or more complex than the original software, needs to be designed and created before the original problem can be solved.

Software solutions can be difficult to find, and criteria for evaluating or comparing solutions (assuming a software team can practically explore different options) are seldom certain. Some attempts have been made to create metrics for objectively measuring programming attributes (e.g., complexity), and some techniques exist for evaluating user interfaces (especially their usability) and software in its totality (especially robustness), but these are far from a complete answer to the problem of evaluating software solutions. Additionally, such techniques are not regularly applied. Generally, the thrust in the software industry has been less focused on measuring software's attributes and more toward creating software that meets its requirements. These requirements span not only social requirements, as mentioned above, but also technical requirements (e.g., performance measures) and system requirements (e.g., compatibility with other devices or software), and any of these may change during the course of a project. Software solutions can be difficult to find, and it can be difficult to know when a solution has been achieved because it is hard to measure the adequacy of solutions.

Wicked problems are also challenging because of the difficulty in working them out through representational means (i.e., artefacts). Brooks (1987) points out that all visual representations of programs are inadequate because they use a spatial metaphor to represent software, which does not occupy space. For the same reason, there is a problem identifying software parts that are useful abstractions (e.g., languages, parts libraries, etc.) because there are endless possibilities for defining parts. Analogously, all static visual representations such as drawings are inherently inadequate in capturing an end user's experience with software because static visualizations do not portray a critical dimension of the design—namely, changes over time. As a result, team members find

communicating through visual representations of programs or user interfaces to be of limited effectiveness.

This section highlighted some of the unusual aspects of software as an object and software-creation work. Uniqueness of the software object, historically speaking, was identified as one reason for difficulties experienced in software-creation work. Other difficulties identified related to factors that make software development a wicked problem, and still others related to problems with representing software to make intermediary work visible.

Certain aspects of this challenging or unusual work are the responsibility of the developer, such as managing the complexity of the software and grappling with the technological environment, but other aspects of this work are rapidly becoming the responsibility of the designer, such as ensuring the future software fits its intended social milieu. Both experts can therefore be seen as having inherently unique and challenging work. The point was made that software objects are not like products produced through industrial design work in many regards. The next sections look more closely at the disciplines within which the role of the developer and then the designer are debated and determined.

Disciplines Addressing Programs and Programming

The Nature of Software Development Work

Software development work emerged in the 1950s and 1960s, but its history is not well known. Thus far, most of what has counted as history in the area of computing has had to do with the history of machines, great individuals, “firsts” or, more recently, the history of specific categories of software (e.g., database management systems). These are the interests of the IEEE Annals of the History of Computing, the Virtual Museum of

Computing (Bowen, 2006) and the International Conference on the History of Computing (Mahoney, 2000). Few conferences, workshops, and seminars have addressed the topic of software creation from a historical or cultural perspective. For example, since the year 2000 researchers from the Software History Center (Grad & Johnson, 2002) have aimed to “preserve the history of the software industry,” but few researchers have looked at the discipline of software creation from a sociological or cultural perspective.

Still, some research has been conducted on the history of software development. The earliest interest in a history of software’s disciplines was expressed by Hamming who, in 1967, “pleaded for a history of computing that pursued the contextual development of ideas” (as cited in Mahoney, 1990, p.325). He was the first to suggest that a history of computing should include the places, people, and especially the situations (i.e., the conferences, workshops, organizations, and so on) in which ideas about computing emerged and developed. In 2004, Nathan Ensmenger (2004), a historian and sociologist suggested a change of focus amongst computer historians from machines to people and from technological developments to social processes and “how things came to be” in the field of computing. Similarly, Mahoney declared his interest in the history of computer science and software engineering (i.e., applied computer science) as disciplines, the development of software, software itself, and software in use; for this reason, he is referenced frequently here.

Looking at software engineering as a discipline, Mahoney (2002) has suggested that a historical account could look at software engineering’s changing *agenda*:

what its [the discipline’s] practitioners agreed ought to be done, a consensus concerning the problems of the field, their order of importance or priority, the means of solving them (the tools of the trade), and perhaps, most importantly, what constitutes a solution.” (p. 10)

Looking at practitioners, professionals, and academics in terms of their common agenda helps to identify the object of a discipline's activity and affords insight into the purpose or motives of a discipline's activity. Mahoney was interested in tools and the developers' object (the software itself) because "the tools and techniques of a field embody its agenda" (Mahoney, 2002, p. 25). As he says,

We [Historians] have to construct what they [groups historians study such as computer scientists or software engineers] thought from the evidence of what they did, and that work ... has often involved a certain amount of speculation. ... This is all the more the case in the history of technology, where up to the present century the inventor and engineer have * as Derek Price once put it * 'thought with their fingertips', leaving the record of their thinking in the artefacts they have designed rather than in texts they have written. (markings Mahoney's) (Mahoney, 1990, p. 325)

This approach suggests that there is more to be discerned from a study of software artefacts, namely the socio-cultural history, than simply the software's design.

Mahoney (2005) states that the processes influencing the development of software have not been studied, and further suggests that many social factors are influential.

However, an influential nonsocial factor he has identified involves the various attempts made by developers to model software solutions using artefacts because these representations have repercussions in real life interactions (Mahoney, 2005). So, he sees two interesting avenues to pursue to study the processes influencing the development of software. One is the history of social forces that have shaped the software that was developed and the other is a history of developers' modeling tools and their role in supporting interactions within software teams. The next subsection pursues the first topic of forces, as revealed through disturbances in disciplinary dialogues about software development. The last section pursues the latter topic of software tools in real life interactions. In this latter section, tools are not just construed as software tools (e.g.,

compilers, drawing tools, and so on), but also include *methods*, meaning proposed processes for creating software.

Science in the service of software development. Programming is a difficult task. Brooks (1987) addressed additional challenges that relate to the nature of software-creation work itself, and not just the nature of software problems. He says software development is inherently complex because no two parts are alike because whenever there are parts that are alike, a developer will generally build an abstraction (one part) to be used in place of the two repeated parts that existed before. Software entities, he says, are “more complex for their size than perhaps any other human construct.” (p. 11)

For many involved in the discipline of computer science a common agenda has been to seek a science that would help to explain or organize the complex activity of software creation. Although the science of software should be as much social as it is intellectual if software-creation work is viewed as a wicked problem, software and software development have been largely construed as intellectual problem-solving activities (Mahoney, 2002). Mahoney claims that the discipline of software development has deliberately positioned itself as a scientific enterprise that is theoretically grounded because this has ensured prosperity and growth through access to resources from the government and the research industry. However, as he points out, there has been difficulty in identifying which science to apply to software development.

For example, John McCarthy had the idea that he would use mathematics to reason about programming and that this could be its scientific foundation. He used lambda calculus to describe the formal semantics of a program (its meaning), which helped him to prove certain programs correct in a mathematical sense if their assumptions could be specified (Mahoney, 2002). His rigorous methods, and others like it, eventually

helped computer science to achieve standing as a separate, but still scientific, discipline based on mathematics.

However, there has been some doubt that mathematics is an effective scientific basis for software development. Mahoney (2002) reports that a committee set up in 1979 (The Computer Science and Engineering Research Study) concluded that computer science (the theory) seemed to follow developments in software development (the practice) and was therefore aiming at “a moving target” (p. 39). One problem has been that while those inside the discipline were always making progress in appropriating mathematical techniques to develop software, the problem of scaling up to the level required by practice was regularly out of reach of these proposed methods. Glass (1996) cites many examples of practice preceding theory, particularly in the areas of software design, software maintenance, modeling, simulation and metrics. Therefore, some claim, if scientific theory (i.e., reasoning about software using mathematics) is following software development, even if only in some areas (Glass, 1996; Mahoney, 2002), it is difficult to see how it can simultaneously be a foundational science upon which software is developed. That is, mathematics as a science of software development is inadequate to the task.

Even Hoare, who would be considered one of the strongest supporters of the idea that software development is mathematical science, has made reference to the strained relationship between mathematics and software development. In 1985 he claimed there were four “principles of computation”: “computers are mathematical machines, programs are mathematical expressions, programming language is mathematical theory and programming is mathematical activity” (Mahoney, 2002, p. 34). However, he also declared that these principles were “held as self-evident, which is just as well, because all

the actual evidence is against them” (Mahoney, 2002, p. 35). Hoare’s point was that software was inherently mathematical in theory, despite the reality of programs that were not mathematical expressions, languages that were not theoretically based, etc.

Despite this tension, Hoare and others were very influential in the field of computing for many years and, as a result, a unique view of theory in the field of computer science developed—theory came to be seen as mathematics. From this perspective, computer scientists, in applying theory, developed mathematical proofs of program complexity, computability or correctness etc. Thus, *theory* or *theoretical foundation* has come to have a special meaning within computer science. It is not theory in the sense that facts or phenomena are explained, as in psychology or other natural sciences; it is mathematics, most usually in the form of proofs based on mathematical assumptions about software. All of this has created a significant gap between what computer scientists learn at university (i.e., the agendas of their discipline) and what programmers do.

However, in Mahoney’s (2004) opinion, computer science, the discipline where software developers emerged, took the form of “a family of loosely related research agendas,” (p. 8) the use of lambda calculus being only one example, and that the applicability of the theory to the practice was always a significant issue both for those inside and outside the discipline. In Mahoney’s view the purported science behind software development was actually disparate “collections of knowledge, techniques and ideas” (i.e., no cohesive science at all). In this view, theoretical computer science, software development’s theoretical foundation, was influenced by a number of different fields from which it appropriated tools. And yet Mahoney also claims that mathematics is not without value, stating that the layers of software closest to the machine may in fact be

mathematical, but that higher-level layers, such as the user interface, are more socially constituted and therefore based on social sciences.

Simon (1996) says that software itself represents a new kind of science, a “science of the artificial,” and Mahoney (1988) suggested that “unlike the extremes, the middle [the software domain] remains a craft, technical rather than technological, [and] mathematical only in appearance” (p. 115). For Mahoney, the science of software is “a science of a world of our own making” (p. 26):

There is nothing natural about software or any science of software. Programs exist only because we write them, we write them only because we have built computers on which to run them, and the programs we write ultimately reflect the structures of those computers. Computers are artefacts, programs are artefacts, and models of the world created by programs are artefacts. Hence, any science about any of these must be a science of a world of our own making... (p. 26).

Certainly if one considers science to be the study of natural phenomena through application of the scientific method, there is hardly any connection between computing and science. However, more intriguing is the suggestion that software may be based on a science of virtual or imaginary worlds. I believe that the “world” Mahoney referred to is the world the users enter when they interact with software, and which they must understand in terms of its possibilities and limitations. Effectively, this is the world that the designer (and sometimes, by default, the developer) creates through the user interface. This world has been variously understood as a human-machine dialogue, a dialogue with the designer through the software, or in the gaming industry, a gaming environment. However, another imaginary world is the programming world of the developer, which also has its own possibilities, limitations, and laws. But, reasoning about such imaginary worlds and developing a science for them, though intriguing, is not a well-developed approach in the disciplines.

Postmodernism and software development. Some researchers have suggested that to this point software development has been viewed as a rational/intellectual activity (Robinson, Hall, Hovenden, & Rachel, 1998). This modernist perspective, which assumes a French mid-19th century progressive view of life that embraces change within disciplines (i.e., especially with regard to technique and theory), is said to have penetrated the disciplines of software creation. According to Robinson et al. (1998) modernism has permeated the computing industry and has not only contributed to the way software is created, it has shaped the identities of developers with qualities valued by modernism, such as being rational and coherent.

Echoing modernism's values, modern accounts of the history of the software discipline have captured change in a linear, coherent, rational, and authoritative narrative. For example, Shapiro's (1997) description of software development's history focuses on the development of issues in the discipline such as complexity, quality, the value of formal methods, languages, life cycles, and development environments. For each of these issues Shapiro claims that developers have had to compromise between generality (i.e., producing ideas that are broadly applicable) and specificity (i.e., producing ideas that address a greatly reduced class of problems).

Another example is Harrison's history of the development of software developers' tools and development environments (i.e., sets of tools). This work showed that software developers have been active toolmakers and that the object of their activity has both broadened from a singular focus on programming to the development of tools to support the management, creation and maintenance of software. Further, it claims to show that programming itself has developed as an activity into the designing of programs (Harrison, Ossher, & Tarr, 2000).

A final example is Raccoon's (1997) account of software history where *tides* are the central organizing element. He proposes six of these, each one representing "simultaneous interest in a group of different technologies, or generations of technologies ... that share the same zeitgeist (spirit of the times)" (p. 89). Writing in 1996, Raccoon identified the naïve tide (1945–1955), the functions tide (1956–1966), the structured programming tide (1967–1977), the modules tide (1978–1988), the objects tide, i.e., a unit of software that bundles together code and data (1989–1996 and beyond), and correctly predicted a patterned programming tide (beginning from about 2000 onwards). Raccoon also claimed tides influenced the way developers thought.

This reported orderliness in the development of software creation activities may be changing as a result of postmodern influences. Huyssen, as cited by Robinson et al. (1998), says postmodernism can be captured by a set of qualities that distinguish it from modernism: "indeterminacies, fragmentation, decanonization, self-less-ness and depth-less-ness, [the] unrepresentable and unrepresentable, irony, hybridization, carnivalization, performance and participation, constructionism and immanence" (p. 368). Robinson et al. claim postmodern views of software development are not only possible, there is already evidence to suggest that development work is acquiring postmodern qualities. This essentially proposes that a view is developing that suggests that software development is not only a rational activity.

From a postmodern perspective, far from being a rational, engineering-like activity that produces robust software, software creation is beginning to be seen as, and is becoming, "... an amalgam of various analytical, design, implementation, predictive and managerial activities, embedded in dynamic social systems, replete with already-developed sites of cooperation and conflict" (Robinson et al., 1998, p. 368). According to

Robinson et al., software development, particularly in the guise of software engineering, is steeped in a modernist tradition and that this (i.e., its modernist and false view of itself) is its major problem. To support their claim that software engineering is steeped in modernism, Robinson et al. argue that software engineers value reason, rationality, control and progress toward an ideal; they engage in a never-ending search for *silver bullets*, and constantly push for progress and control of processes—all of which reveals modernistic assumptions (Robinson et al., 1998). However, these researchers also describe fragments of evidence dating from the late 1990s indicating that modernistic assumptions within the field of software development are beginning to break down. As examples, they cite a move away from structured methods, the serious doubts raised about the value of formal methods, questions asked about whether software development should be underpinned by a mathematical science, and recent expansion within the field of software engineering to technological and human concerns. All of these changes open up the possibility for the recognition of significantly more indeterminacies that require more post-modern, rather than modern, skill sets. Robinson's account, taken in its totality, suggests that the whole movement to see developers as software engineers may, in fact, be the story of the influence of modernism on software development.

Noble and Biddle (Biddle & Martin, 2003; Noble and Biddle, 2002; 2004; 2006) also take a postmodern perspective. Citing a variety of evidence Noble and Biddle claim that postmodernism is manifesting itself within software development despite any conscious awareness of this influence by those within the discipline. For example, they have observed that software is no longer programmed from scratch, but made up of heterogeneous fragments of variously produced interacting parts. Similarly, programming languages have changed; older languages such as Pascal and C supported programming

from scratch, but newer languages such as Perl and C++ provide more support for connecting together disparate software elements . Further, they say that representations such as object-oriented modelling techniques (where object here refers to a unit of software that bundles together code and data) originally used solely to model real-world entities or objects as software objects on a one-to-one basis, are now being repurposed to “propose a range of different descriptions of the relationship between programmatic object and external object” (Noble & Biddle, 2002, p. 51). They also cite a shift in views on requirements specification. “Post-modern computer science holds that no requirements [specifications] can be both complete and consistent: you have to pick one” (Noble & Biddle, 2002, p. 52). Modern approaches to requirements specification, in contrast, always favoured consistency because of the belief that it was possible to produce neat, ideal, elegant, abstract specifications. A postmodern approach leans more toward completeness and attempts to match requirements to messy real world requirements, rather than idealizing them and representing them as they should be.

Still, with all of these developments, Noble and Biddle (2002) point out that the postmodern approach does not replace or eschew modernism, rather “post-modernism admits modernism as one mode of expression” (p. 59). In fact, Biddle has suggested that the move to postmodernism exposes and resolves the contradiction between software development as a mathematical activity and software development as an activity with no scientific underpinnings, by suggesting that it has multiple scientific underpinnings. In Noble and Biddle’s view, in place of software developers seeking a rational solution to software problems, they see software-creation work becoming a transformed eclectic activity—a patchwork application of programming languages, methods, and techniques. Along with this, they have also re-envisioned computer science as a discipline that no

longer relies so heavily on mathematics, but one that becomes an “unrestricted science, from where investigators must be prepared to follow their problems into any other science whatsoever” (Noble & Biddle, 2002, p. 60).

Experts in Software Development Work

Having examined some key disturbances and contradictions within software disciplines on the nature of software development work, this analysis now turns to look more closely at the disturbances found in debates about the role of the developer.

Software developers as software engineers. The term *software engineer*, used to refer to a software developer, became popular as a result of a 1968 conference in Garmisch (Mahoney, 2004; Randell, 1979). The proceedings of this conference state that “the phrase ‘software engineering’ was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering” (Naur & Randell, 1994, p.88). Other terms also emerged out of the conference, including cold-war inspired rhetoric such as *software crisis* (which refers especially to the problem of increasing software complexity due to developments in technology) and *software gap*, which refers to the difference between the capability of the hardware and software (Randell, 1979; 1996). Such talk justified the need for software engineers and shaped the agenda of the following decades.

At the time of the 1968 conference there were about 100 different programming languages, but only three were widely used (Sammet, 1969) and only about 500 companies employed people to write programs. System software, such as operating systems, was still in its infancy and the engineering-backed ideas of software modularity and software structure were just beginning to be explored. In Mahoney’s (Mahoney,

1990, p. 325) view, in 1968 “programs became ‘software’” (p. 329) and the developer’s task became more distanced from the computer itself. This occurred because a body of programs had begun to amass making computers useful for more complex tasks than mere mathematical computations, and a body of people had begun to emerge who were declaring themselves to be experts in the production of programs. In essence, a social system was emerging complete with a basic tool set, a clearly defined object (i.e., software), a promising market, and an emerging new role.

The 1968 conference was a turning point because the place of software development in the larger world of computing and the marketplace was addressed. At that time, software was becoming a commodity in its own right, distinct from computers (Randell, 1979). The conference also addressed the issue of creating software. One participant, Stanley Gill, was quoted as saying, “Software is as vital as hardware, and in many cases much more complex, but it is much less well understood. It is a new branch of engineering” (Randell, 1979, p. 7). Randell (1979) reports there was also much talk of design and how to do it best. For example, Peter Naur described how the task of software production could be sequenced; Merritt Roe Smith bemoaned the fact that many of the steps in the process he followed produced “nothing but paper” (p. 8); and Douglas McIlroy spoke about mass-producing software. Mahoney (1990) reports the outcome of the conference was to set an agenda. The phrase *software engineering* was used at the conference as an expression of a requirement or a wish for how software should be produced, rather than a phrase that described a healthy, well-developed discipline (Mahoney, 1990; Randell, 1979, p. 7).

The term software engineering is still ill-defined because “behind the concept of software engineering lie models drawn from other realms of engineering, often without

much reflection on their applicability” (Mahoney, 1990, p. 334). For instance, mechanical engineering seemed like a sensible model to apply to software development because both practices are about designing and constructing. Talk of subassemblies, interchangeable parts, tools, and the like, that is regularly heard in software circles, originates from this analogy (McIlroy, 1976). Another model, industrial engineering, proposed by Bauer in 1971 (Mahoney, 1990), emphasized organizational and managerial tools, ideas that originated with F. W. Taylor and Henry Ford, and likened software *production* to the processes that might occur in a production plant. Out of this position emerged terms and ideas like structured analysis, structured design and structured programming (Mahoney, 1990).

To make progress in understanding how to develop software, early developers saw their task as engineering. However, Mahoney (Mahoney, 1990) believes the search for a software engineering model has been futile because traditional engineers can apply known and tested principles on a much more regular basis, in comparison to software developers whose work is more innovative and disconnected from its theoretical foundations. More recently, software development has been likened to architectural work. Regardless of which analogy is used, any position that is taken potentially influences the general approach to practice. For example, those with an engineering perspective may tend to work more from specifications or tend to test or measure software more because these tasks are more in line with those of engineers. In contrast, an architectural approach draws attention to the building of the product and the elements used in the building of the product. For example, in the past decade, the patterns approach of Christopher Alexander (1977; 1979) was enthusiastically adopted and applied to software construction.

Software developers as professionals. Whether or not software development is engineering, the question arises, is it a profession—a field that has a comprehensive body of knowledge, professional associations, codes of conduct, accreditation processes, licenses and certification standards? Social scientist Nathan Ensmenger (2001) identifies the 1950s and 1960s as a defining period where basic tensions regarding whether developers were professionals or not were rooted. In the 1950s, programming was viewed as a creative activity and eccentricity was a desirable attribute of a programmer. The thrust to establish computing as a discipline was begun by the Association of Computing Machinery (ACM) in 1959 when the association actively sought a cohesive and consistent theory or body of theories (Ensmenger, 2001 p. 62). Computer specialists supported these efforts for the benefits it would bring them, and managers supported it to the extent that it would provide them with a “standardized, traceable workforce,” although “professional efforts that encouraged elitism, protectionism, or anything that smacked of unionism were seen as counterproductive” (Ensmenger, 2001, p. 69). In the 1960s, computer specialists felt threatened because of movements to automate their work, and they responded by becoming more professional to get “increased social status, greater autonomy, and improved opportunities for advancement, and better pay” according to Harold Wilensky (1964, as cited in Ensmenger, 2001, p. 61).

Today developers are typically expected to have memberships, certificates, diplomas or degrees from government- or industry-sanctioned institutions that are highly regarded professional organizations. Beyond this, there are a plethora of opportunities to acquire further professional accreditations, which attest to a developer’s knowledge or skills. Some authorities consider these essential to the field (P. Kruchten, 2010). Certifications are mainly indicators of a developer’s practical experience, but when

accompanied by training to achieve the certification, can also be a way for a developer to move into a different area of practice. Some certifications are associated with vendors and a specific tool or technology such as Microsoft and its development platforms. Others are associated with general professional organizations, such as the Canadian Information Processing Society (CIPS). For the software developer, the problem is to select appropriate certifications that reflect his or her skills and to keep up in their rapidly developing field.

Beyond the issue of certifications, there are issues of licensing that have been substantially more contentious. Licensing is an authorization from an authority, which allows the holder to do certain things that are restricted to protect the public. Issues of licensing for developers who are software engineers have been dealt with differently in different countries. In Canada in the mid-1990s Memorial University in Newfoundland was sued by provincial and national engineering associations for using the term *engineering* to describe a programme in software development. The lawsuit was discontinued in 1999 when a compromise was reached allowing the Council of Universities and Colleges to control the use of this term and to ensure that it was not misused (Connors, 1999). Today, Canadian software developers may become engineers and subsequently licensed professionals if their university's programme is certified (Engineers Canada, 2004) and they gain relevant experience. It is also possible to become a licensed engineer if one's background is in computer science, a process that was initiated by the Professionals Engineers of Ontario in 2001. In actuality, most developers are neither engineers nor licensed, and the debate continues over issues of public safety and the identification of exactly which software practices need to be licensed

(Professional Engineers of Ontario, 2004; Rowlands, 1999). Canadian views represent a moderate perspective on licensing relative to others countries.

In Britain the British Computer Society (BCS) (British Computer Society, 2006a) and the Engineering Council of the United Kingdom (Engineering council UK, 2006) collaborated to develop a set of examinations administered by the Information Society Examination Board (ISEB) (British Computer Society, 2006c). Universities seek accreditation from the British Computer Society to exempt their students from ISEB exams. IT professionals work toward certificates and qualifications in a complicated system of examinations and chartered statuses. The net effect is that Britain's developers are typically also licensed engineers (British Computer Society, 2006b).

In the United States, developers are generally not licensed engineers. In the 1990s, the National Society of Professional Engineers sued all states in America to prohibit the use of the term "software engineer," winning in most states except Texas, which issues licenses to software engineers (Wikinfo, 2006), although now the National Society of Professional Engineers is moving steadily toward licensing (National Society of Professional Engineers, 2006).

Internationally there is strong support for identifying developers as software engineers within the Institute for Electrical and Electronics Engineers (IEEE), a prestigious American-based organization. In 1990 they partnered with the Association of Computer Machinery (ACM) and formed a committee called SWECC (the software engineering coordinating committee) to explore the possibility of making software engineering a profession. However, ACM pulled out of this work in 1998, partly because of a lack of a standard body of knowledge for the discipline, and partly because it had some practical objections to the licensing of software engineers based on ACM's

perception of licensing's effectiveness for ensuring software quality and reliability (Allen et al., 1999). IEEE then went on to create and publish SWEBOK, the SoftWare Engineering Body of Knowledge (Barbacci et al., 2005), which has had a large influence in the production of textbooks, courses, and degrees and the adoption of an engineering outlook to software development work.

With respect to the professional status of developers, the response to the issue of professionalism has varied from country to country and has even resulted in major disagreements at the international level, showing that the depiction of developers as software engineers is a socially constructed, contentious role.

Contradictions in the Developer's Role

In the section on developers, I have reported on disturbances and contradictions within the discipline of computer science and its offspring software engineering. A picture emerged of a relatively new discipline with the challenging problem of creating software, which has struggled to define its agenda and form an identity for itself and its practitioners. Of particular historical significance was the emergence of a practically-oriented, engineering-influenced discipline—software engineering—a field that began from within computer science, but which has since been aggressively challenged from both inside and outside engineering circles.

I next report on the disturbances and contradictions that are evident at the discipline level from the literature review. Then I report on my reflections on how contradictions at the disciplinary level were reflected or evident in contradictions in the role of the developer.

I worked back and forth between disturbances and contradictions that I found in the literature review, and matched each contradiction to multiple disturbances.

Occasionally, I needed to articulate a new contradiction for a set of disturbances. In the end, I created a table consisting of six contradictions, each associated with multiple disturbances. In columns 1 and 2 of Table 9, I list the relations between contradictions and specific disturbances. In this table and in subsequent similar tables, I use these symbols “◀▶” to clearly express a contradiction and the forces/ideas that are in tension with each other.

Table 9 *Contradictions in the Developer's Role*

Disciplinary contradictions	Disciplinary disturbances	Role contradiction
Software is a uniquely new object, so software-creation work is new, ◀▶ yet software-creation work is considered to be like the work of other disciplines.	<ul style="list-style-type: none"> ▪ Attempts to compare the field of software to other sciences, engineering, or architecture ▪ Challenges tracking software and applying intellectual property protection and other laws to software. ▪ Unique delivery and maintenance mechanisms for software 	A developer may consider him or herself to be like a worker in another discipline, ◀▶ but this analogy will necessarily have limitations.
Software is invisible and intangible, ◀▶ yet has visible and tangible effects (Mahoney, 2000, p. 285).	<ul style="list-style-type: none"> ▪ Software cannot be fully represented in spatial dimensions so some aspects are inevitably ignored 	A developer needs visualizations to carry out his role, ◀▶ but none are fully adequate.
Creating software is inherently difficult, ◀▶ yet the expectation is that the work can be prescribed.	<ul style="list-style-type: none"> ▪ Finding software solutions is hard ▪ Many different well-defined steps to solving problems have been proposed, but not applied in practice 	A developer's methods for solving software problems are not without value, ◀▶ but they do not solve the problem of creating the software.
A science of software is a "world of our own making," ◀▶ yet existing theories from other sciences, especially mathematics, are applied to the task of understanding software (Mahoney, 2002).	<ul style="list-style-type: none"> ▪ Attempts to use mathematical theory to aid software development proves feasible on small problems, but challenging to scale upwards 	A developer does not have a science to help him or her solve problems, ◀▶ but rather draws from multiple sciences or no science.
Software-creation work is not like engineering because its object is immaterial and its object does not have a science, ◀▶ yet software-creation work, like engineering, is about building software to specifications to solve practical problems.	<ul style="list-style-type: none"> ▪ Software engineering as a discipline is unwelcome in many countries, but welcome in others ▪ Committee concludes computer science followed developments in software engineering ▪ Gap between mathematical theories and practice of software engineering 	A developer's role may be to build software to specification, ◀▶ but their ability to do this in all matters is limited.
Software-creation work is technical (about coding) and therefore a rational/intellectual organized activity of one or more people, ◀▶ yet software-creation work is also technological (about hardware), and social (about social systems) and is therefore an indeterminate, fragmented, eclectic activity. (Noble & Biddle, 2002; Robinson et al., 1998).	<ul style="list-style-type: none"> ▪ Solutions reveal more problems ▪ Requirements are contradictory, changing, and even <i>un-specifiable</i> ▪ Completed software frequently does not fit requirements ▪ Value of reason, rationality, control, progress to an ideal within discipline is at odds with indeterminate reality software problems ▪ Advocating structure and formalisms as complete solutions is at odds with reality 	A developer's primary task (coding) is highly rational, ◀▶ but other aspects of his or her task cannot be approached using reason alone.

The next step of the analysis was to clarify the implications of the contradiction for the role of the developer. This had the result of revealing nuances and tensions in the role that would have been missed had the role been described solely by its function. Column 3 in Table 9 lists contradictions in the role of the developer that were derived by reflecting on column 1 and, to a lesser extent, column 2.

For example, in row 1 column 2 several disturbances are listed that all relate to the unique nature of software work. These include (a) attempts to compare software development to work in other sciences, to engineering disciplines, or architecture; (b) challenges tracking software and applying intellectual property protection and other laws to software; and (c) unique delivery and maintenance mechanisms for software.

The contradiction that explains these disturbances in row 1 column 1 is “software is a uniquely new object, so software-creation work is new, yet software-creation work is considered to be like the work of other disciplines.” Comparisons with work in other disciplines have repeatedly been shown to be of limited value, and attempting to treat software as though it was like other objects, such as those that are industrially designed, has strained many institutions (e.g., patent offices, government privacy laws, etc.) that have attempted to treat software similarly to objects that are designed then produced.

In row 1, column 3 the corresponding tension in the role of the developer is articulated. The assumption here is that the generalized contradiction will also appear in the role of the developer. In this case the contradiction reflected in the role of the developer is that “A developer may consider him or herself to be like a worker in another discipline, ◀▶ but this analogy will necessarily have limitations.” The move from a disciplinary contradiction to the implications for the role of the designer is justified by the assumption that contradictions that are very basic to the activity (i.e., systemic), such as

the ones identified here, will be reflected in other aspects of the system including the role of the developer (i.e., there will be corresponding unresolved tensions in the role). The question I asked myself was “How do developers experience this contradiction in their work?”

Except for changes to the details, the reasoning for subsequent rows is not explained in detail because the reasoning for each row is analogous to the reasoning for row 1.

The most important aspect of the table, given the purpose of this dissertation, is the third column because the text in these cells describes contradictions in the developer’s role. The implications of the disciplinary contradictions for the role are now briefly summarized.

The unusualness of software as an object has implications. A developer may consider him or herself to be like a worker in another discipline, ◀▶ but because of the unusualness of software as an object, this analogy will necessarily have limitations.

The amorphous nature of software has consequences. A developer needs visualizations to carry out his role, ◀▶ but because of the nature of software, none are fully adequate.

Together the uniqueness and amorphous nature contribute to the inherently challenging aspect of a software development work. A developer’s methods for solving software problems are not without value, ◀▶ but because software is inherently difficult, they do not solve the problem of creating the software.

To address the challenging nature of software development work, the search for a theory to support the activity has been ongoing. A developer does not have a science to help him or her solve problems, ◀▶ but because multiple sciences have been explored

for their applicability, developers may draw from these to aid their work or, failing this, they may draw from nonscientific fields or no specific disciplinary fields.

Because the science has been illusive, developers who consider themselves to be engineers are faced with a dilemma in terms of foundational knowledge. A developer's role may be to build software to specification, ◀▶ but because the software object is very unusual for an engineer, their ability to do this in all matters is limited.

And finally, the skills required of the software developer are changing rapidly as software penetrates ever more deeply into our lives. A developer's primary task (coding) is highly rational, ◀▶ but because of the increasingly diverse aspects of software-creation work, other aspects of his or her task cannot be approached using reason alone.

All of this points to an essential contradiction in software development work. Essentially, the developer's work is highly creative because it involves the creation of new ideas, concepts and associations. But, arriving at software solutions is challenging work, and the results of this creativity are invisible, difficult to assess, and unrecognized because the software object is amorphous. Instead, most understandings of the developer's work attempt to see it as a productive activity that can be understood and prescribed, and whose solutions can scale up to an industrial level in a way similar to products that are designed via an industrial design process.

To summarize, this analysis helped to create an understanding of the developer's role. By focusing on contradictions at the discipline level, the original conceptualization of the developer as the person responsible for the production of the code was significantly expanded. This analysis connected contradictions inherent in the discipline with conflicts inherent in the developer's role and thereby created a more nuanced view of this role.

Disciplines Addressing User Interfaces and Designing

In order to better understand what it is like to be a designer, this section looks at the disturbances and contradictions found within the disciplines from which designers emerge.

Throughout this section I use the term human-computer interaction (HCI) to refer to all the disparate disciplines from which designers emerge, although there is no agreement on a term for this. In fact, one of the major points of this section is that many disciplines contribute knowledge and expertise to the topic of designing software user interfaces. Moreover, there is no consensus on what these disciplines might be, let alone what all of them altogether might be called. Because of this enormous breadth and the lack of clear boundaries, this analysis emphasizes a part of HCI that most specifically attempts to address the breadth of disciplinary contributions. The Special Interest Group on HCI (SIGCHI) is a subgroup of the Association of Computing Machinery (ACM), a large educational, scientific, and international computing society—the same association mentioned in the previous section on developers.

Unlike the developers' discipline, no historians or social scientists have examined the development of the disciplines of designers, so this analysis relies more heavily on the reflections of designers who have been a part of designer disciplinary dialogues for some time.

The Nature of Design Work

Design's rapidly evolving object. The object of user interaction design research and practice has changed rapidly over the decades. Keeping up as a researcher or as a practitioner on issues related to user interaction has been challenging. This is because all aspects of interactive systems have evolved rapidly, including computing hardware,

computing software, the makeup of the end-user communities, and the context in which interactive systems are used. This section briefly describes these changes using the research of the Computer History Museum², a museum in Mountain View, California that maintains both a brick-and-mortar and an online presence. However, the information from this virtual museum has been reinterpreted to emphasize the particular aspects of interactive systems mentioned above: (a) the computer technology or hardware through which the user interface is experienced, (b) the software platforms that support the development of interface software, (c) the user base that uses interactive software, and (d) the contexts in which interactive software is used. In the course of this reinterpretation, a number of attributions are mentioned that are supported by the best information available to the Computer History Museum, but some of these details are still debatable. The point here is not to highlight debatable issues or argue these details, but to support the claim that change has been rapid and to provide a clearer sense of the pace of this change. To this end, this review begins in the 1940s when HCI human factors and ergonomics researchers were beginning to run the very earliest studies of computer operators. Throughout this review I make very little mention of the disciplines that attended to and often formed around the topic of computer usage, leaving these details for the next section on the proliferation of disciplines.

In the early 1940s computers were special-purpose machines based on vacuum tubes or relays that were operated by trained operators and used by governments for largely military purposes. Computers were programmed and used through awkward means such as building new parts, direct manipulation of the hardware, or by punching

² <http://www.computerhistory.org/timeline/>

holes in paper tape. The user interface was the machine itself, and the end users were typically the same people who built or maintained these machines. This changed in 1945 when John von Neumann proposed building stored-program computers so specialists could control computers by writing programs and running them in the computer's memory. Work immediately began on building this type of machine. In 1949 Maurice Wilkes built the first practical stored-program computer for use at Cambridge University—this computer's program was stored on paper tape. Wilkes' accomplishment realized Turing's 1947 goal of building a general-purpose computer that would "do all sorts of different things simply by programming rather than by the addition of extra apparatus," according to the Computer History Museum.

In 1950 the first commercial computer was built in the United States and sold to the U.S. Navy. Shortly thereafter, England's first commercial computer was built to solve the problem of daily scheduling production and delivery of cakes to the Lyons teashops, a very early application of computers for business purposes. The American government bought a computer for the Census Bureau and used another to successfully predict election results from early return data. The machines were sometimes laboriously programmed by writing code in assembly language that was very close to the machine hardware. Later in that decade Grace Hopper developed the first compiler, which meant programmers could use English-language constructs to program. In that era, end users were typically experts in their own fields and also experts on the computers they were using. In the mid-1950s, technologies evolved further, especially because the invention of transistors made computers both faster and more reliable. IBM and other manufacturers began building and selling machines on a broader scale. The first operating systems appeared, and also better programming languages, transforming the work of programmers

so that they could focus solely on the software without concerning themselves with the machine's hardware. Computers began to be used for scientific purposes, in manufacturing and in banking. In just 10 years computers, system software, users, and the contexts in which computers were used had transformed significantly.

The early 1960s saw the invention of integrated circuits on chips and the development and sale of general-purpose mainframes and minicomputers. These were entering universities where academics and students applied them to new areas, including music and video games. At the same time, computers were installed in production plants and began to be used by factory workers. Early computer hobbyists were bringing them into their homes. But more than anything, computers were entering businesses where they were used to process business transactions. During this era, programmers were the de facto designers for all of these applications. When hardware and system software continued to improve, this presented new possibilities. Timesharing was introduced to allow programs to be run simultaneously and interactively. Structured programming techniques allowed more complex programs to be developed, and advances were made in operating systems, further easing the programming task. Hardware engineers began developing graphical devices, and communication networks were adapted to link computers together.

In the 1970s, all the developments in the 1960s became widely applied. Computers were built with microprocessors reducing costs significantly. Graphics software for games and animation systems was developed. E-mail began to be used and standards were developed for networking. Leveraging many of these developments together, a revolutionary prototype for end-user computing was designed based on psychological principles about how humans think, learn, and act—the Alto computer had

a mouse, a primitive windowing system, advanced graphic capabilities, and could be networked. In 1977, several companies released the very earliest personal computers. Computers were being widely used in many industries, in academia, and now in homes.

In the 1980s, improved technologies for personal computing began to be widely applied for interactive applications. Interface designers were creating graphical user interfaces, and novel modes of interaction began to be practical, including musical interfaces, speech recognition, information visualization, and virtual reality applications. Together these transformed whole kinds of work, such as graphic art and publishing.

In the 1990s, many network technologies reached maturity and widespread application, leading to the World Wide Web and mobile computing, creating the possibility of diverse application areas—especially ones that were no longer tied to physical locations, such as E-Commerce. This resulted in an even broader range of end users and even more widespread adoption of computer technology. At the same time, special purpose software systems for developing user interfaces were becoming increasingly popular.

The concept of an interactive system changed rapidly with each advancing decade, a claim supported in this dissertation by describing changes in various aspects of such systems. Grudin (1990) captured all of these aspects in his analysis of the evolving interface. “The location of the ‘user interface’,” he says, “has been pushed farther and farther out from the computer itself, [and] deeper into the user and the work environment” (p. 261). He described how the concept of the human interface was initially referred to as hardware in the 1950s, then software applications in the 1960s, then the *terminal* in the 1970s, the “dialogue between human and computers” in the 1980s, and finally in the 1990s to “that which is between the workplace and humans.”

Similarly, Battarbee (2004) captured this change in its totality by describing the evolution of the concerns of researchers and practitioners with respect to the significant attributes of the user interface. Desirable attributes, he says, developed from initial concerns about the interface being functional, to useful, to usable, to becoming, most recently, a collection of qualities that humans may experience when interacting with computers, such as satisfaction, pleasure, challenge, excitement, aesthetic appeal, etc.

Other researchers who have captured this change in its totality are Myers and colleagues (2000), who report that researchers (themselves included) and developers have had trouble developing the necessary platforms and tools to build new interfaces. They provide evidence of how specialized tools for designing the interface and platforms for developing the user interface, which are now used ubiquitously, have been challenging to create. Developers, they say, have struggled to create the right tools because “of an increasing diversity of user interfaces [demanded by new social circumstances] on an increasing diversity of computerized devices (i.e., new technologies)” (p. 3).

The object of design research and practice has been a rapidly moving target. This has resulted in a number of disturbances, including an evolving definition of the concept of the user interface, an expansion in the understanding of the qualities of an interface, and challenges delivering platforms and tools for developing such interfaces.

A proliferation of supporting disciplines, forums and communities. The last section provided a sense of the context in which designers and practitioners work. This section addresses the challenging task of identifying the mass of disciplines and organizations that became engaged in the issues associated with creating interactive systems and are involved in the multidisciplinary field of HCI.

Interest in HCI began in the 1940s when human factors and ergonomics researchers studied the operators who laboriously maintained expensive vacuum-tube or relay-built computers in running order (Dix, Finlay, Abowd, & Beale, 1998). This work was an extension of work on man-machine systems in general. In fact, human factors and ergonomics was largely the only group engaged in HCI until computers began to be purchased and used by businesses, which drew the interests of the information systems community who were already addressing problems related to information flow in business environments. Interface design for office systems became a research stream in information systems in about 1967 (Grudin, 2005). Then, in the mid-1970s, the University of Cambridge's Unit for Research in Applied Psychology, which had been engaged in human factors work for some time, began to address the problems of work that involved computers, and for many years produced significant research in this area.

However, disciplinary involvement in the field really began to change in the late 1970s. In 1978 a Special Interest Group on Social and Behavioral Computing (SIGSOC), a small but enthusiastic group of people within ACM, declared an interest in studying the relationship between people and machines. This group's interest went beyond workers in the workplace, and the group was later to become SIGCHI, the Special Interest Group in Computer-Human Interaction and a centre of activity in the field of HCI.

At the time they declared their new agenda, SIGSOC's membership was comprised of people who were involved in social and behavioural sciences who were using computers in their research, but whose primary interest was not the use of computers. The first conference organized by SIGSOC was held in May of 1981, ostensibly the first multidisciplinary HCI conference ever. However, despite the conference, the membership of SIGSOC failed to grow, the name of the group being

identified as one problem. Shortly after a second conference in 1982, ACM changed SIGSOC's name to SIGCHI. SIGCHI's first newsletter, the SIGCHI Bulletin recorded that SIGCHI would address the following issues: "Human factors in the interaction process, monitoring and evaluation of human performance, design alternatives directed toward the improvement of the human interface, and methods of improving communication between computer scientists, behavioural scientists and the end-users (sic)" (Borman, 1996, p. 1). However, between the proposal by SIGSOC in 1978 and the creation of SIGCHI in 1983, the social scientists had left the group. Borman, who was a leader in SIGSOC and in SIGCHI in the early days explains this by saying,

As computing became more pervasive, especially in universities and research centres, other groups evolved which focused even more specifically on the concerns of the social scientists. ... it became apparent that that constituency were [sic] more comfortable in their own professional associations, rather than in the ACM. (Borman, 1996, p. 1)

And, although the human factors and ergonomics community apparently attended the first SIGCHI conference, they also did not return to subsequent conferences. One exception, human factors specialist, Stuart Card who remained, made this comment about efforts to transform his own discipline to be less "second-class," "Hard science, in the form of engineering [meaning computer science and software engineering], drives out soft science, in the form of human factors" (1985, as cited in Grudin, 2005, p. 51). So, in the very early days of this important group, for whatever reasons, certain groups stopped participating. This left the computer scientists, software engineers, cognitive psychologists, and various researchers in human behaviour as SIGCHI organizers and attendees. SIGCHI computer scientists built prototypes of novel user interfaces or devices, and SIGCHI psychologists provided behavioural or cognitive information

regarding human capabilities with graphical user interfaces (GUIs), which were new at the time (Card, Newell, & Moran, 1983).

Three years after SIGCHI formed, social scientists organized a separate conference on Computer Supported Collaborative Work (CSCW), which was sponsored jointly by SIGCHI and a Special Interest Group on Supporting Group Work (SIGGROUP). Many social scientists found a venue for their research in this conference, which addressed issues to do with technology and groups, organizations, communities, or societies. This conference now runs every second year, and its counterpart, the European Computer Supported Collaborative Work conference (ECSCW), runs in alternate years.

SIGCHI's interest in interactive systems distinguished itself from the other disciplines in HCI at the time because of its interest in discretionary users of computers. This meant that a whole host of new issues arose, such as the usability and learnability of interactive systems. The SIGCHI community reformed around these new issues and around new technologies and attracted more researchers from other disciplines. For example, in the 1980s the advent of educational software for children encouraged developmental psychologists and training experts to become involved in HCI research. Then, in the 1990s media people, graphic designers, industrial designers, filmmakers, narratologists, sociologists, anthropologists, and dramaturgists became engaged as SIGCHI continued to adapt and expand, as it continues to do to this day.

SIGCHI is now a significant professional organization in HCI, but it is not the only one. Other organizations include the Usability Professionals Association (UPA) established in 1991, and an association of interaction designers called IxDA formed in 2003. Aside from these specialized organizations, broader groups such as a number of design disciplines, information systems, and human factors and ergonomics groups have

subgroups dedicated to HCI. An example is the Computer Systems Technical Group of the Human Factors and Ergonomics Society. All of these groups are distinguished largely by their primary foci. The human factors and ergonomics groups organize around the concept of “efficiency,” the information systems people around “information flow,” the SIGCHI and Interact groups around “the user and usability,” the CSCW attendees around “user-groups,” and the interaction design group around “design practice and experience,” although there are overlaps. Also, each group is proficient in different methods, leverages different sets of research, and advocates different values, the net effect is that differences between groups can be large.

Today designers typically identify with either a specific discipline, professional group or one of the following communities: human factors and ergonomics; information systems; SIGCHI (developers, psychologists, designers, practitioners and others); CSCW (developers, social scientists and others); IFIP TC13 INTERACT (the International Federation for Information Processing) (Lindgaard, 2006); Interaction (an international conference like CHI); interaction design; or any number of other groups that organize around specific technologies, products, or markets. These latter groups can be very large because of the importance of industries such as Web design and computer games. Rather than one discipline, HCI involves multiple, loosely linked disciplines with increasingly diverse views on roles, their objects, theories, tools, methods, and standards.

Disturbances arise as a result of the distinct cultures of the multiple disciplines participating in HCI. Carroll (2003b) says that “practitioners ... often manage the intellectual scope of concepts and approaches by deliberately isolating themselves from some portion of the field’s foundations” (p. 6). Also, even long-standing members of the field have trouble making sense of it all. For instance, there are various ways of seeing

the relations between these groups and which one encompasses others. Sharp, Rogers and Preece (2002) see various academic disciplines (e.g., psychology and cognitive science), interdisciplinary fields (i.e., HCI, CSCW, human factors and information systems), and design practices (e.g., graphic design and product design) feeding into a larger field called Interaction Design. Grudin (2005) sees HCI as a blend of human factors, information systems, and the disciplines participating in SIGCHI (largely computer scientists, psychologists, and designers).

Within groups there is little coherence. As Grudin (2004) says, “If we think of CSCW as an emerging field or common enterprise, we may be frustrated by the mosaic of different pieces, the frequent misunderstandings, and the lack of intellectual coherence” (p. 26). Carroll (2009) describes the HCI community as a collection of disciplines with no real agreement on concepts, methods, problems, assumptions, or applications.

Disturbances are evidenced in differing values placed on conference versus journal publications versus portfolios (Grudin, 2004). Others are found in problematic terminology. For example, Grudin (2005) found that on the surface, the discipline of human factors and ergonomics shares many terms with the SIGCHI and information systems communities, but on closer examination he also found that many of these terms had significantly different meanings. Also, he found that even within a single community there are problems with terminology such as the SIGCHI community’s use of the terms *user interface*, *user*, and *designer*, which originated from one perspective (software engineering), but now obscures how interfaces are developed and used for the entire design community (Grudin, 1993).

With such a diverse discipline, an individual's involvement with the various HCI communities influences how designers identify themselves and see their role (Brown, 2005). Practical design activity is situated within this broad framework of diverse, but related, systems of disciplines and organizations. Disturbances include challenges with conceptualizing the entire field, designers isolating themselves from parts of it, and interdisciplinary tensions.

The search for theoretical foundations to support the work. While many feel HCI should be a purely applied discipline, others yearn for a more scientific, theoretically-based discipline. Although early work evaluating software began 70 years ago by researchers who studied human interaction with machines of any type, HCI as a discipline with a focus on human-computer interaction really only began 30 years ago. The discipline's very earliest form involved two disciplines that, according to Carroll (2003a, p. 1), were based neither on science nor theory: human factors engineering and software engineering.

Soon after the emergence of HCI as a broad discipline, various cognitive approaches were explored to understand and conceptualize interaction with the aim of addressing the issue of how to design software. In 1985, Newell's address to SIGCHI conference attendees shifted the direction of the discipline when he described HCI as an emerging *scientific* discipline and posited cognitive psychology as the theoretical basis from which HCI research and practice could proceed (Card et al., 1983). In Carroll's (2003a) view, the 1980s was the era when cognitive science dominated, calling this time a brief, but "golden age," of focused activity—because the domain of users was clearly identifiable, user tasks, purpose and environment were equally clear, and only one theory was being used to address a commonly agreed upon problem. However, cognitive

approaches had their limitations as a source of theory and the days when cognition was the sole theory for HCI passed quickly. The SIGCHI community began a search for other theoretical foundations.

One of many examples of theoretical foundations is Bødker's (Bødker, 1991) work using activity theory. According to Carroll (2003a), Bødker's work "integrates the analysis of individual behaviour and experience with interpersonal cooperation (including division of labour) and culture (including tools and socially developed practices)" (p. 5) and puts the emphasis on user goals in the context of broader motivations, human development, and social systems (Brown & Strickland, 2005b; 2005a; 2007).

Another example, which serves to represent the diversity of theoretical foundations explored, was Laurel's (1991) work. In her approach, the interaction between the human and computer is understood as dramatic activity, and therefore the task of the designer is to create imaginary worlds where end users extend their natural abilities to think, feel and act. Using ideas from Aristotle and the ancient Greek theatre, she developed an approach to design that leveraged drama theory.

Many such potentially foundational theories were explored. By the mid-1990s, HCI drew from theories in many social, psychological and behavioural sciences, as well as various arts and design disciplines, and, as a consequence, a tremendous range of empirical methods and scientific and nonscientific concepts were routinely used to study and design interactive systems.

In 2003, Carroll (2003a) suggested there were too many theories, methods, application domains, and systems, and that these were threatening to break the discipline apart and render it ineffectual for practicing designers. To justify this he said he had

observed that most practitioners and many researchers felt their work was most productive when they ignored theory and worked purely from a design perspective.

A number of individuals have attempted to resolve the tension between theory and practice. Sharp et al. (2002), taking a conciliatory position, say that HCI has roots in theory, but relies strongly on good practices or experiences to actually create usable products. Shneiderman sees psychology's theory and methods as being potentially very useful in practice (Shneiderman & Plaisant, 2004). Dix et al. (1998) posit that HCI may be comparable to architecture, a marriage of art/craft and science. This debate has been prolonged. As far back as 1986, Norman and Draper (1986) asserted that all theories were inadequate in a sense, even though each field makes a contribution toward an understanding of interaction.

A user interface functions at the intersection of many different kinds of things: people, machines, tasks, groups of people, groups of machines, and more. These different things contribute diverse constraints and opportunities to the design process. None of them, not even machines, is captured by adequate theories, that is theories adequate to tell the designer what's going to happen when a design is put to work (Norman & Draper, 1986, p. 5)

The field of HCI is diverse and complex. The disruptions reviewed here include expressions of confusion over what the field is, the threat of the SIGCHI community (the most diverse of all HCI communities) breaking up, reports of ineffectual theories, and the attempts to come to terms with the relationship between theory and practice.

Experts in Design Work

The designer's role: diversity of viewpoints. Within such a diverse discipline, the development of the designer's role (i.e., the understanding of the designer's object and their orientation to that object) has taken pluralistic directions. Because HCI has no standardized curriculum, this section reviews books written by a representative sample of

well-known people in HCI. Many of these books are introductory textbooks in university HCI courses. In the descriptions below I outline each expert's views on the role and tasks of designers and their theoretical foundations or justification for these views. The point of this review is not to be exhaustive, but to illustrate the diversity of training an HCI specialist may receive.

Shneiderman (1987), a computer scientist, published the first version of his book *Designing for the User Interface: Strategies for Effective Human-Computer Interaction* in 1987. Republished repeatedly since then, the 2004 version of this book (Shneiderman & Plaisant, 2004) suggested the user interface designer's concerns should be focused on measurable user interface issues such as time to learn, speed of performance, rate of errors by users, retention over time, and satisfaction. It proposed that indicators for these measures be used to drive the design and that psychologists also use these same measures to develop a theoretical foundation to guide the work of designers. This book advises on screen layout, the selection of user interface styles (menu driven versus form-fill-in, etc.), and the arrangement of large amounts of information on a screen. It suggests the use of various tools for specifying interfaces and building prototypes. This text was, and still is, a popular choice for many university HCI courses.

Greenbaum (an information systems scholar) and Kyng (a computer scientist) (1991) published the edited collection *Design at work: Cooperative design of computer systems* in 1991, which emerged out of design experiences in Denmark and Sweden. The book's central premise, that design is situated, led to the suggestion that software products should be designed with the active participation of end users in their work context. Participatory design, the name given to this approach, encourages designers to be aware of culture, politics, and economics in the workplace. Design is accomplished

through a series of workshops with end users and other stakeholders. In workshops participants use materials such as prototypes of products, or electronic games that stimulate discussion and ideas, or whatever seems appropriateness for the design situation. Designers act as facilitators, initiate the workshops, and conduct research on behalf of the workshop participants. According to the authors this approach is informed by the humanities, social science, and computer science, although they do not say how. Such a book would comfortably find a home in graduate courses in HCI.

Jakob Nielsen (1994), a computer scientist, published *Usability Engineering* in 1994, which focused on the achievement of usability goals when designing, these being learnability, efficiency, memorability, reduced error rates, and increased satisfaction. Nielsen argued that project managers should realize that 48% of all code is dedicated to implementing the user interface, so mistakes made in designing the user interface may be costly to repair (Nielsen, 1994, p. ix). He also stressed that products with a well-designed interface save companies money because employees are not wasting time fighting with software. Nielsen's usability engineering lifecycle has fourteen steps and culminates in the design of a usable product through a process that emphasizes prototyping and measuring progress to goals. Artefacts used by usability practitioners include informal scenarios, user-interface mock-ups, and the like. The designer's work is completed after a prototype meets product-specific standards for usability. Nielsen offered a practical methodology based on a business perspective. This approach to design originated with Deborah Mayhew in the late 1980s, was popularized by Nielsen in the 1990s, and has since been extended by others. Many designers would be familiar with at least one of the interpretations of usability engineering either through a course, exposure through a professional organization, or because they read about it.

Carroll (1995), a cognitive psychologist, published the edited collection *Scenario-Based Design: Envisioning Work and Technology in System Development* in 1995, a book about leveraging people's natural story-telling abilities. Contributing authors, such as Rosson, pointed out how scenarios (i.e., stories) are good at raising questions, envisioning new activities, and evaluating ideas. The book suggests beginning the design task by focusing on activities that need to be supported and not on software product requirements. Designers write stories of people using existing systems and then they write stories about people using envisioned of future systems. They capture their design rationale (the reasons for particular design decisions) in "design claims." The steps are (a) designing an activity by understanding the users' motives and tasks; (b) designing the information structure; (c) envisioning sequences of actions; (d) prototyping; (e) usability evaluation; and lastly, (f) documentation. The theoretical foundations for this approach are varied, which reflects the diversity of the contributing authors' theoretical backgrounds. Exposure to this approach would typically occur in graduate courses in HCI.

Sharp (a computer scientist), Rogers (a cognitive scientist), and Preece (an information systems scholar) (2002) published *Interaction Design: Beyond Human-Computer Interaction* in 2002. Designing, they say, should involve anyone who has a stake in the software product throughout; usability and user experience goals should guide the project's progress; and designers should expect to reiterate through four steps, namely (a) identifying needs and establishing requirements; (b) developing alternative designs; (c) building interactive versions; and (d) evaluating prototypes. Their process aims to create applications that are useful, usable, safe, satisfying, enjoyable, fun, entertaining, helpful, motivating, aesthetically pleasing, supportive of creativity,

rewarding, and emotionally fulfilling. The approach is based on a collection of theoretical frameworks about cognition, such as mental models and information processing models, computational offloading, etc. Design artefacts may include prototyping tools, but they also suggest using video techniques to understand what product to build. This text is a popular choice for undergraduate course in HCI.

This nonexhaustive review of some of the popular or well-known texts used in undergraduate and graduate HCI courses reflects the diversity in the approaches taken to teaching design and justifying it. As a result of designers' diverse educational backgrounds, there is no singular way to be a designer and this is reflected in a number of disturbances. For example, a study of six designers I conducted in 2005 revealed that designers believe others do not understand their practice, that they are not comfortable with all of the expectations placed on them, and that they are somewhat bewildered about their role (Brown, 2005). This paper also exposed the diversity of titles used to describe the HCI specialist's role and the diversity of terms used by designers themselves to describe their role. These ranged from usability specialist to interaction designer to user interface designer, and so on. Many of these of course refer to specializations within the field, but the boundaries between these are not clearly distinguishable.

This review omitted many, many approaches, such as those based on theories of graphic design or industrial design. However, enough diversity was considered to explain some of the disturbances that appear in the lives of design practitioners, such as the problems designers have because others do not understand their field, that designers are sometimes expected to do certain tasks that they are not well trained for, that designers are not always sure what they should be doing, and that there is a proliferation of terms to

refer to designers, such as user interface designer, interaction designer, user experience designer and so on.

Designers as professionals. The presence of coherent organizations and their initiatives is indicative of progress toward forming a profession. One kind of initiative involves the development of significant conferences, of which ACM's SIGCHI and IFIP's INTERACT are examples. Another is the creation of a curriculum. A curriculum was completed in 1992 by the ACM SIGCHI Curriculum Development Committee³ (Hewett et al., 1992, p. 5). This committee generated exemplars for courses and saw their work fitting into the ACM/IEEE curriculum for computer science, information systems programs, or interdisciplinary, undergraduate specializations in HCI; however, although used by computer scientists, it is far from being uniformly adopted.

More recently, some commercial and professional organizations in the field of HCI have begun offering certification programs and certificates to designers, but none of these have become de facto standards. For example, Human Factors Interactional (HFI) offers certification as a usability analyst. But other organizations such as the Usability Professionals' Association (UPA) decided in 2002 not to offer certifications because it would be "premature" (UPA Board of Directors, 2002); today it still does not offer certification programs.

Other indicators of progress toward professional status include the publication of de facto guidelines for design, such as the once very popular, but now dated, Smith and Mosier *Guidelines for Designing User Interface Software* (S. L. Smith & Mosier, 1986)

³ <http://old.sigchi.org/cdg/>

or more recent and rapidly evolving standards on Web accessibility put out by the World Wide Web Consortium (W3C).

Many disturbances are evidenced when considering issues of professionalism and design. Certification programs are disputed and those that exist are not standardized and not well recognized. The proposed curriculum is used mainly in computer science departments, but not in other academic disciplines, and guidelines become rapidly dated.

Contradictions in the Designer's Role

This section began by seeking disturbances within the disciplines of designers. A picture emerged of HCI as a very new, very broad, and weakly defined field involving collections of disciplines, organizations, forums, and communities that bring together researchers and practitioners. This review revealed many disturbances and from these a list of contradictions was created, as shown in columns 1 and 2 of Table 10.

As with the previous analysis, after identifying contradictions in the field, these contradictions were transformed to express how they are reflected in the role of the designer. Column 3 in Table 10 lists these role contradictions, which are summarized below.

Table 10 *Contradictions in the Designer's Role*

Disciplinary contradictions	Disciplinary disturbances	Role contradiction
Large investments are made to design user interfaces, ◀▶ yet technology, system software, users and contexts of use change rapidly forcing continuous redesign work for the design team	<ul style="list-style-type: none"> ▪ The <i>interface</i> as a moving target (Grudin, 1990) ▪ Desirable qualities of interfaces evolving constantly (Battarbee, 2004) ▪ There is a struggle to build specialized tools for designing and implementing the user interface that keeps up with demands (Myers et al., 2000) 	A designer expects to design novel interfaces, ◀▶ but much of his or her career is spent redesigning and adapting interfaces to new devices, platforms, user groups or environments.
User interface design is one topic, ◀▶ yet many diverse disciplines lay claim to this one topic.	<ul style="list-style-type: none"> ▪ There are different ways of conceptualizing the field, ▪ Little coherence is shown within disciplinary groups, ▪ Interdisciplinary rivalry persists, frequent misunderstandings, ▪ Intellectual coherence is lacking ▪ Differing values remain about success ▪ Misunderstandings continue about terms between disciplines 	A designer will take a disciplinary-based perspective to their work, ◀▶ but they may not feel a connection to the wider field.
A user interface functions at the intersection of many different kinds of things which contribute diverse constraints and opportunities, ◀▶ yet none of them is captured by adequate theories that will predict what will happen when the design is put to work (Norman & Draper, 1986)	<ul style="list-style-type: none"> ▪ There are fears of the discipline breaking up ▪ Too many theories are discussed rendering HCI ineffectual for practicing designers. ▪ Practitioners can be most productive when they ignore theory and work purely from a design perspective. ▪ There are many ways that theory is seen in the discipline (as roots, as potentially useful, as complementary to the art/craft of design work) 	A designer may use theories explicitly or implicitly to guide their work, ◀▶ but disparate theories may not lead to a unified design and nothing will ever replace trying and seeing what happens.
Practical, theoretically founded approaches to design are valuable, ◀▶ but in practical work situations designers must adapt to how others work.	<ul style="list-style-type: none"> ▪ Nondesigners misunderstand design practice ▪ Designers are not comfortable with all expectations placed upon them ▪ Designers have many titles ▪ Certification standards are ad hoc ▪ No curriculum is adopted uniformly 	A designer may have an idea about how they should design, ◀▶ but may find it challenging to apply or adapt these approaches within the multidisciplinary teams they find themselves.

The object of design activity evolves rapidly. A designer expects to design novel interfaces, ◀▶ but because changes to technology, system software, users and contexts of use are so rapid, much of a designer's career is spent redesigning and adapting interfaces to new devices, platforms, user groups or environments.

The expansive nature of the discipline of HCI has implications. A designer will take a disciplinary-based perspective to their work, ◀▶ but because many disciplines lay claim to this field, designers may not feel a connection to the wider field of HCI generally.

Many theories have been applied in designing. In fact, a designer may use theories explicitly or implicitly to guide their work, ◀▶ but because there are so many disconnected theories, they may find that disparate theories do not lead to a unified design and nothing will ever replace trying and seeing what happens (i.e., prototyping) in design work.

Many good proposals have been put forth for how to design. As a result, a designer may have strong views about how they should design, ◀▶ but because the designer necessarily works in a multidisciplinary team, they may find it challenging to apply these approaches within the multidisciplinary teams they find themselves.

Approaches to the Integration of Design and Development Work

Software was developed for decades without the help of user interface designers because early input and output was very primitive; early computer users were either developers themselves or specially trained, and applications were in-house. The introduction of designers into software and product development processes or the melding of design and development processes represents a disruption, historically

speaking. This section details various approaches to coordinating the work of designers and developers, beginning with a short review of some evidence of a disruption.

In 1997 I surveyed software engineering and HCI design processes by reviewing books undergraduates might read—especially textbooks (Brown, 1997). I paid particular attention to descriptions of processes that described the rules about how design or development work should ideally be conducted. These processes were valuable artefacts for clarifying roles and for breaking a very large and complex task down into a number of stages, yet I noticed there were very few connections made between the processes of one field with the processes of the other. For example, one of the major software engineering texts (Pfleeger, 1998) made no mention of interface design. Another hinted that software engineering and design processes should be mixed at appropriate points (Pressman, 1992), but provided no details. I found only one book for developers that proposed a way of combining processes from the two fields (Downton, 1993). This book suggested that methods from HCI could be inserted into the steps of a classic software development lifecycle called the “waterfall method.”

At the same time, a number of HCI processes were also being described in readily available books; however, they did not specifically address the role of the software developer’s process relative to the HCI specialist’s process and they were not being used in computer science or software engineering courses. Examples of such books included a number of those mentioned in the section on designers (Carroll, 1995; Greenbaum & Kyng, 1991; Nielsen, 1994). Problems integrating the work of the disciplines that were revealed by inspecting texts included differing views on the importance of lifecycles, differing vocabularies, and differing views on methods (i.e., software engineers preferred semiformal methods and designers preferred informal methods). In this paper (Brown,

1997) I concluded that because the methods were so disjointed, they were inadequate to the task of clarifying roles and responsibilities, structuring project work, and assessing the state of a project in real practice.

In practice, there were also problems when the two disciplines worked together. As evidence, in 2003 an IFIP group launched a series of workshops called Bridging the SE and HCI Communities, which were held at various conferences from 2003 to 2005 (Harning, 2004). In launching the series, IFIP referred to barriers in education, processes, methods and vocabulary, and stressed that software engineers and HCI specialists needed to collaborate more effectively, given that as much as half of the software for interactive applications was dedicated to implementing the user interface. And, because this required significant programming effort by developers, designer errors had serious project consequences. Of course, other issues relating to requirements also suggested the need for better collaboration.

John looked at common processes of HCI specialists (ISO 13407) and software engineers (the Rational Unified Process) for an explanation (John, Bass, & Adams, 2003) and found that the differences between processes were telling. For instance, the designer's process had a step for addressing the product in use, recording design rationale, and for checking the competency of the staff and methods, whereas the developers' process did not. In contrast, the developer's process addressed business issues, suggested performing a risk analyses, and addressed issues to do with the software architecture and with deliverables, whereas the designer's process did not. John concluded that the HCI community and software engineering community were not aligned in their concerns and that a number of items that could potentially be shared were not. She felt each community would benefit from expanding its concerns.

A number of researchers attempted to analyze the problem. For example, Seffah and Metzker (2004) believed that one barrier to integration was that HCI techniques were hard for developers to learn and apply. A second problem was the illusive concept of usability, which was confusing to developers. Another problem was that HCI techniques were not part of the software development lifecycle, and therefore not easily integrated into the project plans. Seffah and Metzker suggested that the HCI specialist's limited focus and toolset was a roadblock to joint activity. They also found that both professions lacked knowledge of the other discipline. This work showed that appropriation of HCI techniques by developers or the introduction of an HCI specialist onto a project team were both problematic.

Integration of Designer and Developer Processes

One possibility for integrating work practices is that software engineers use the techniques developed by HCI specialists. A rigorous approach to this was suggested by Coutaz (1995), who noted that usability was generally ignored in software projects. She showed how usability techniques have a “natural anchor point” in the V model, a software engineering process that emphasizes software testing throughout the development cycle. Applying usability techniques during development, she felt, would reduce the costs of correcting usability errors after a release. She adopted an engineering perspective to integrating design and development work through her focus on document enhancements and by suggesting that developers adopt the more structured techniques from HCI. She suggested adding task analysis from HCI to the software developer's requirements step and that the model produced by executing these steps be incorporated into the software engineering requirements document. She suggested the quality plan document be augmented with usability goals and that these should be tracked. She also

suggested using an HCI notation for specifying user interfaces called the user action notation (UAN) in the system design step and evaluation steps.

Another attempt at appropriating techniques was suggested more recently by Ferre, Juristo and Moreno (2004). They conducted an extensive survey of methods using textbooks in the field of HCI, looking for techniques that developers could use. Their review showed that many modern HCI processes overlapped significantly and differed only in terms of emphasis. They suggested a new software development process based on SWEBOK (the ACM standard process prescribed for software engineers) with what they considered to be the best HCI techniques slotted in at appropriate points in the process.

Seffah, Djouab and Antunes (2001) also began with the ISO 13407 standard and the Unified Process. Noting that both processes were document-centric (or in ISO's case, form-centric), they searched for common representations that could be used in either set of documents. They suggested that *use cases*, which are short, structured stories of software use and system responses, could be a key artefact, arguing that this form of requirements specification was understandable to both practitioners. The researchers proposed a new process called Approach Centered on Usability and Driven by Use Cases (ACUDUC) that blended usability and use issues into the software engineering requirements phase. They also explored making designer and developer artefacts useful to others outside the profession and making connections between artefacts. For instance, they showed the points of connection between the task analysis diagrams created by HCI specialists and the use case diagrams created by developers to show the relationship between user tasks, user actions, and system responses. As well as making suggested changes to processes and tools they also suggested that the roles of the various specialists

be clarified, with usability engineers focused on context of use and developers focused on functionality.

Sousa, Furtado, and Mendoca (2005) proposed Upi, a lightweight development process, for developing interactive systems inspired by HCI specialists Constantine and Lockwood's (2003) Usage Centered Design Rules that incorporated best practices from RUP (The Rational Unified Process). RUP (P. Kruchten, 2004) is considered a fairly comprehensive process in software engineering in that it can scale down or up and requires very few or very many documents and diagrams. This work began with processes and practices that were well-known, which were then integrated them with the goal of improving usability and productivity. There are 14 steps in this proposed integrated process starting from eliciting stakeholder needs and finding actors and use cases, and ending with the deployment and evaluation of the entire software system. Sousa et al. also suggested developing an integrated set of software tools that would allow designers and developers to work with a rich collection of artefacts. Their process was trialed in industry and was reportedly a big improvement over a previously used process that neglected interface design issues entirely.

Sousa et al. (2005) see improvements occurring through the creation of new integrated processes and new tools, but they also see a need to develop a common objective or motive to unite the disciplines. They suggest that the common motive of designers and software engineers should be a software system that is designed, built, and deployed by the end of the project.

Integration Through Linked Designer-Developer Artefacts

Other researchers have addressed the integration problem by suggesting changes to tools rather than processes. My research in the late 1990s and early 2000s with

Nicholas Graham took this approach (Brown & Marshall, 1998; Brown & Johnston, 2001b; 2001a; Brown, 2003). I developed a prototype of a tool for software engineers and HCI developers assuming that artefacts were central to software-creation work. I hypothesized that an integrated toolkit that allowed designers to see relations between artefacts would help bridge the gap between the disciplines because the linked artefacts would show how the work in one discipline related to the work in the other discipline. Graham developed light-weight linking algorithms that aimed to be good enough, rather than perfect. The idea was to allow the designer or developer to work separately, but to also allow them to see the relations between their work and the work of the other whenever they required. One innovative feature of this tool was that it did not require that artefacts in one discipline be consistent with artefacts in another discipline.

An assumption of this work was that artefacts would co-evolve (Fisher, Arias, Eden, Gorman, & Scharff, 2001) and that access to the artefacts created by practitioners in another discipline would support this natural process. As a result of being able to see connections between one's work and the work of another's through artefacts, we expected each specialist's artefacts would change. The software developer's artefacts would change as a consequence of the developer developing a deepened understanding of user needs as expressed in the designer's artefacts. The designer's artefacts would change in response to the designer developing an increased awareness of difficulties encountered by the developers in building the corresponding software for their designs. For HCI specialists, linking artefacts would reveal the complexity of the task in front of the developers and would help them have an idea of how much of the software was implemented or what parts were being worked on. For the developers, access to HCI

artefacts would help them understand and respect what the designer did and remind them of the perspective of the end user and the context of use.

Initially, a simple collection of four artefacts were linked together by hard-coding the links in a tool called Vista (Brown & Marshall, 1998). These were the scenario and task analysis diagrams produced by many designers, and the software architecture diagrams and programs produced by software engineers. The task analysis and architecture diagrams were linkable because each represented an abstract view of the interaction and the software respectively, whereas the scenarios and code were linkable because they were both artefacts closely related to the concrete actions of the user.

In another paper my colleagues and I (Brown, Graham, & Wright, 1998) addressed cultural and technical problems associated with linking these artefacts. We explored a number of techniques that would surmount problems encountered in linking, such as the mismatching of names between two diagrams that happened when the designer gave an element, such as a button, one name and the software engineers gave that same button a slightly different name. Graham (Brown et al., 1998) showed that with a very small set of rules, a considerable amount of linking could be done semiautomatically.

In 2001, a commonly used artefact called a prototype was added to the mix of artefacts already identified and linked in a tool called ScenicVista (Brown & Johnston, 2001b; 2001a). At the time, the prototype was seen as a central shared artefact because of its meaning for all stakeholders on the design team, including designers, developers, project managers, and users. Everyone, it seemed, could relate to the prototype, making it the ideal central artefact from which others could be linked. Our prototype was interactive and was linked to other artefacts because it was specified both visually and

textually using an emerging XML standard for structuring data. The links were made to the textual description; however, the designers and developers saw links to the visual representation, which was more meaningful. One of the beneficial consequences of linking artefacts was that a person could step through one artefact (i.e., explore it systematically), such as the steps within a scenario, and automatically see the prototype step through the corresponding screen actions (buttons highlighting, menus appearing, and so on). ScenicVista displayed the appropriate screen for each step in the scenario and highlighted elements in the prototype that were relevant to the action in the scenario, such as a certain button highlighting when the step in a scenario was “click a certain button.” Users could control which artefacts they saw and how much screen space they used. When a user indicated an interest in an element of a diagram by clicking on it, all corresponding elements in other diagrams were highlighted so viewers could quickly identify them, thereby supporting designers and developers to see connections between their work (Brown, 2003). A similar system, referencing the work on Vista, as the first system of this type, was recently described by Memmel, Vanderdockt, and Reiterer (Mommel, Vanderdonckt, & Reiterer, 2008). Their goal was to support HCI specialists, software engineers, and business process analysts to design the user interface through the use of a shared tool, and a small informal empirical study produced very positive feedback.

Other researchers have also explored integration from an artefact-centric perspective. Van Harmelen (2001), in his edited collection *Object Modeling and User Interface Design* proposed connections at the object-modeling level (object here means a unit of software that bundles together code and data) with HCI artefacts such as scenarios, tasks, and use cases. Van Harmelen thought that simplified object models

could link to scenarios, task models, and domain models. He suggested HCI professionals learn object-modelling techniques to make them more compatible with their software engineering colleagues.

In the same vein, Constantine and Lockwood (2001) described how to use “essential use cases” to guide user interface design. Use cases were artefacts that were being used in the software engineering community to specify the interaction between the user and the system and to show the consequential interaction between system objects. However, Constantine’s essential use cases, which were abstract, structured descriptions of humans and computers interacting, made them more useful for exploring various user interface designs. Their approach focused on usage and not users, making “supporting the work of the users” the central concern. It made model-making the dominant design activity (there are models for user roles, tasks and interface content) and suggested designers be involved in software-creation work prior to developers. However, they also thought that involvement throughout the development process would work, especially when designers melded with developers using Agile methods (Constantine & Lockwood, 2003).

Rosson and Carroll (2001) suggested that scenarios were the key artefact that could help with the construction of object models. The advantage of the scenario, they claimed, was that it kept the focus on concrete activity and away from abstractions of the work, which is something developers are not prone to doing because they are experts in abstract thinking. Scenarios are very specific, detailed, step-by-step accounts of how an interaction would occur.

Using semiotics theory, de Paula, Barbos, and de Lucena (2005) recently proposed a new artefact based on their knowledge about conversations. The motive for

their work was their dissatisfaction with all other artefacts, which they claimed were at the wrong level of abstraction to support communication about requirements (a central concern of both designers and developers). They hoped that their new artefact would help developers and designers understand how their decisions impacted each other's work. Using semiotics they modeled interaction as messages between the user and the designers of the system; the messages comprising a language. This approach put the emphasis on messages (i.e., signs) and on the designer's work directed at conveying meaning through the interface, which de Paula et al. conceptualized as a medium. De Paula et al.'s artefact can be used to generate parts of some Unified Modelling Language (UML) models, which are used by developers to model many different aspects of the software, making a link between a shared designer-developer artefact and other software engineering artefacts. The team developed a tool called MoLIC to model the conversation between the user and the system, and they expressed their intention to link their new artefacts with a range of other developer artefacts.

Integration in Practice

Anecdotal evidence suggests that integrating work processes was not only a concern of academics, but also in industry. Lindgaard, for instance, reported that Telstra Australia had developed an integrated HCI and software engineering process by 1993 (Lindgaard, Havloujian, Chessari, & Bednall, 1993). In her workplace, software engineers were required to meet human factors requirements and guidelines specified by a centralized HCI group. She also reports awareness of similar efforts in British Telecom in the UK as early as 1990. Specialized professional conferences such as Human Factors

in Telecommunications⁴ have published some of these practices at their biannual conferences.

One paper on workplace practices (Anderson, Fleek, Garrity, & Drake, 2001) described the experiences at Siemens' Medical Solutions, where two HCI processes, Contextual Inquiry (Beyer & Holtzblatt, 1997), an HCI approach that emphasizes field studies and modelling, and Constantine and Lockwood's methods, based on essential use cases and modelling, were integrated into RUP, the software engineering process used at Siemens. The problem at the company was that usability was left to the end of the software engineering lifecycle. In deciding to introduce usability practices earlier, they found organizational challenges to be the greatest. Fear of lengthening the development cycle of the developers was a major concern. And, although they began their transformation by focusing on the requirements step, by the end, changes to RUP's elaboration and construction phases were also made. This meant that both the requirements and design phases were altered. The transformation resulted in the creation of new roles at the organization: usability engineer, user interface designer, and usability evaluator.

After studying three companies Radle and Young (Radle & Young, 2001) concluded that there is no one best way of initiating usability into an organization as obstacles and opportunities will vary. In Organization A, the obstacle to integration was a lack of awareness of usability. In that organization change began with the Information Products department manager who believed usability would be critical to their company's success. She had a team of people trained in HCI practices and then had them "infiltrate"

⁴ <http://www.hft.org/>

development teams. Organization B's obstacles were a lack of awareness and a fear of increasing the length of the development cycle. This Fortune 500 company began by creating a Customer Experience Department with highly skilled people. Employees from this department integrated into development groups and became involved in every aspect of the product, including development, marketing, documentation, and technical support. They used standard techniques like usability testing, user profiling, features lists, heuristic evaluations, and competitive evaluations. Organization C began with a group of HCI researchers who were located at the corporation's head office and whose members went into the company's development groups over a ten-year period. In the end, as a result of experience with practice, the HCI researchers developed new tools to support usability work in the development groups and wrote many papers that served to raise awareness among developers at the organization.

A recent emphasis in software development processes has been on *agility*, involving a highly iterative approach to both requirements determination and development, as shown in such methods as Extreme Programming (XP) (Beck & Andres, 2004). From an Agile developer's perspective, Patton (2005) reported that he used task analysis (the top level only) and site visits, both HCI techniques, in a novel way to determine the scope and priorities in an XP project that was so large it needed new methods beyond those commonly used in XP circles. Experienced developers, expert customers and end users were all engaged in the project. This group found that using HCI techniques that focused on usage and context of use helped them decide how to order the development of the software.

Agile methods emphasize the importance of co-located collaboration, and there has been illuminating work showing how developers collaborate and how some novel

artefacts support the process (Sharp, Robinson, & Petre, 2009). However, such studies concentrate on collaboration between developers or with those setting requirements for developers. Further, their focus is on the software engineering of the software objects (i.e., program and code) that provide functionality and a common language for developers (de Souza, Froehlick, & Dourish, 2005; de Souza, Quirk, Trainer, & Redmiles, 2007; de Souza, Sharp, Singer, Cheng, & Venolia, 2009).

Summary

This summary of efforts at integrating software engineering and HCI methods has shown the diversity of approaches taken among academics and practitioners. Much of this work has addressed the creation of new processes that would help specify how work should be conducted when the concerns of both the developers and designers are considered to be important. Other work addressed the identification of joint artefacts or the development of new joint artefacts. Still other efforts approached the task of supporting integration through the development of new tools. In all cases, there was overlap between these efforts (e.g., researchers would start by suggesting a new process and conclude by also suggesting new tools or artefacts or vice versa). These efforts varied in their reliance on theory (most had no theory to guide them) or their connections with practice.

One interesting new approach was the explicit concern for developing a joint motive to unite developers and HCI specialists (Sousa et al., 2005). Other researchers touched on this topic indirectly by developing integrated processes, helping each profession understand the contribution of the other, and by attempting to develop artefacts that would be meaningful to both developers and designers. Some ideas for new shared concerns were the software architecture of the user interface (John et al., 2003)

and plans that would specify the order in which the product should be developed (Patton, 2005).

The review confirmed that disturbances occurred in practice when the work of developers and designers were required for a common project. From these disturbances contradictions related to designer-developer interactions were deduced.

Designer-developer interactions are necessarily ongoing throughout the designing phase. Designers must interact with developers because what is designed must be implementable, ◀▶ yet developers must interact with designers because many designs that can be expressed cannot be feasibly implemented.

The control of the user interface design terrain is disputable. Designers are responsible for the user interface, ◀▶ yet developers also have a legitimate claim to having responsibility for the user interface because developers historically have been designers, not all projects need a designer, and some developers are HCI specialists.

The control of the user interface implementation terrain is also disputable. Developers are responsible for creating the software, ◀▶ yet increasingly, designers produce the software for the user interface automatically through the use of sophisticated drawing tools that produce code automatically, and end users (particularly gamers) are now producing software products unassisted by developers or designers.

A unified process for creating software is not easily arrived at. One would expect that melding one designer process and one developer process would yield a combined designer-developer process that would organize joint work, ◀▶ yet both professions treat their processes as a set of guidelines that can be applied in any order and combining two sets of guidelines does not help order an activity.

Communication through artefacts is problematic in both directions. One would expect that the designer would be able to use artefacts to express the interface that should be built, and that the developer should be able to use artefacts to indicate how they will build it, ◀▶ yet this simple communication is inherently difficult because no artefacts can represent amorphous software, and all spatial artefacts fail to capture the experiential nature of the interface.

Concluding Statements

This analysis looked at the disruptions in the disciplines from which designers and software developers emerged, in order to find disciplinary contradictions. Using the principle of interrelatedness I then showed how these contradictions were reflected in the roles of designers and developers. Both the designer and the developer's roles were shown to be internally contradictory from an activity-theoretical perspective.

Certain aspects of the contradictory nature of the roles related back to a lack of disciplinary coherence. For the developer, one contradiction related to attempts to transfer knowledge from other disciplines to development work. The contradiction exposed tensions developers experience as a result. In particular, a contradiction related to envisioning development work as engineering work was highlighted. Related to this, research exposing the influence of modernism on the developer's discipline predicted a broadening of this search for complementary disciplines into nonscientific domains or possibly a transformation of the search itself. This showed that today modern and postmodern influences in the field of software development present an unacknowledged conflict for the developer.

For the designer, some contradictions inherent in the role relate to the proliferation of disciplines in the field of HCI and the consequential lack of disciplinary

coherence. An inherent contradiction that related to a designer's identification with the field from which they emerged, but a disassociation with other fields, was identified. For both the designer and the developer, lack of disciplinary coherence impacted the practitioner's understanding of their role.

Other contradictions related to the search for theoretical foundations. For the developer, a primary contradiction related to conflicts over the science or sciences to draw on, or whether there was any science to draw on at all. For the designer, the possible use of diverse and not well-integrated theories, which might be explicit or implicit in their work, was contradictory because it led to no assurances of a unified design in the final software. In either case, much discernment was required on the part of the designer and developer with regard to the large gaps between theory and practice.

Both practitioners experienced contradictions related to the abstract nature of their object. The developer, due to software's invisibility, has no choice but to execute their role by using a multiplicity of representations that are transformed by software tools into executable programs. The designer is presented with developing an interactive object, but is often faced with working with artefacts that do not include time as a dimension. In either case, the adequacy of representations used to create software is always an issue.

Both disciplines developed processes that served to break the work of creating software down into ordered parts. While such processes were not without significant value for the developer, there was still a large gap between processes that were developed and the actual steps taken to solve the problem of implementing software, creating tension in the developer's role. For the designer, processes may provide them with an approach to structuring their work, but it may or may not be possible to execute these processes in the context of a multiplicity of conflicting processes adopted by the other

specialists on multidisciplinary software teams. In both cases, processes are a source of both support and conflict.

Both the disciplines and the roles of the designer and developer are internally contradictory. Therefore it is not surprising that work that involves both designers and developers, in the context of multidisciplinary work, is problematic.

Because of the lack of disciplinary coherence inherent in both roles there is no clear boundary between the work of the designer and developer. A developer as software engineer or a developer as architect conceives their role quite differently. This is also true for designers, who may see themselves as leveraging knowledge about drama, narratives or visual design. Because views of tasks vary, actual tasks vary too, and therefore software creation work overlaps in different ways for different pairings of designers and developers, inevitably causing disruptions. This is particularly evident when seen through the research on the development of joint processes. Many processes in use today are one-sided and favour articulating the developer's steps over the designer's or sometimes vice versa. Although many joint processes have been proposed and were presented here, no joint processes are in widespread use.

Communication between designers and developers is also expected to be problematic. For the designers, the multiplicity of representations that the developer works with, and their sheer complexity, makes them inaccessible. Conversely, the developer may experience frustrations when reading the visual representations of the designer because the time dimension, missing from the drawing, is an essential dimension for implementing the behaviour of the software. This literature review featured many examples of researchers developing new artefacts to support this communication work. This work roughly took two tracks. One aimed to develop mutually meaningful artefacts

and the other aimed to develop tools that linked artefacts between disciplines. In either case, the new artefacts aimed to make connections between the work, to solve the problem of communication, and to support collaboration.

One particularly intriguing strand of research aimed to create a new joint object for the combined disciplines. Although such work is still in its infancy, such a direction could potentially transform the work substantially and unite the designer and developer behind a common purpose. From Engeström's perspective, such work represents the highest form of learning and transformation possible (Daniels, 2008a; Engeström, 1987). Engeström's research is particularly sensitive to supporting efforts that reset the direction for joint work. Sousa et al.'s (2005) work on software-creation work suggested that the designer's and developer's work ended only when a software system was successfully deployed, thereby extending both the roles of the designers and developers and the reasons for designer-developer interactions.

This chapter opened with a discussion of the unique nature of the software object and the inherent challenges that this represented for either discipline. All of the searches for disciplinary foundations, theoretical underpinnings, the work on breaking the complex task into steps, and the development of visualizations to aid the work are ample evidence of the complexity of software-creation work. In designer-developer interactions each person may appreciate this fact about their own work, but whether or not they appreciate that the work of the other is inherently difficult is unclear. Many tensions that may arise from this most basic misunderstanding were reviewed in this chapter.

Results and Discussion Perspective 2: Mediated Interactions and Activity

In activity theory, individual action is best understood as part of an activity or activity system because this provides the rationale for actions. The question that motivated this study was: How is the work of designers and developers structured within organizations and what role do artefacts play here? To answer this question, four of the eight designer-developer pairs were observed as they engaged in software development work. Each pair was engaged in a separate software project, and the data collected from these observations were analyzed using activity systems analysis.

The data used in this study were collected as per the description in the method section for Perspective 2 and are summarized in Table 11. The table shows the period, or the number of days between the first and last observation day. Because I entered the workplace on some days and not others, I also show the number of days in the workplace.

Table 11 *Observational Data*

Characteristic	Talia	Jaba	Telespace	Quadrow	Total
Observation period (days)	60	19	8	6	93
Days in workplace	8	6	5	4	23
Observation notes (pages)	61	55	89	81	286
Photos	150	50	125	120	445

Observational notes and the annotated photos that were integrated into these notes were the primary source for this analysis, with interview data collected for Perspective 3 used as a secondary source. Observational notes recorded each goal-oriented action and many goals because these were verbalized as participants worked. However, not all goals were recorded because participants became absorbed in their work, they tired of verbalizing goals, or at times it was simply inappropriate (e.g., in meetings). After meetings, when able, I debriefed with my participants to confirm my understanding of

their goals. Each time an artefact was used, I made a note, and if I did not already have a photo of the artefact, I took one. All photos were annotated on the same day, and then integrated with the notes, which were later typed up.

From these data, and applying activity systems analysis, I identified the activities and the interrelated elements that gave the activities their structure. These were the motives, desired outcomes, objects, objectives, subjects⁵, tools, rules, community, and division of labour established to complete the software projects at the various sites. Activity systems analysis posits that organizing challenging work so as to accomplish it jointly is a natural and common occurrence, and reliance on tools, techniques to divide the labour, etc., is an evolved and now natural human ability, and an appropriate unit for psychological enquiry. This analysis is about how software teams in four organizations brought these natural organizational abilities to bear on the problem of creating new and novel software, but it focuses particularly on how designer-developer interactions advanced software creation work, and particularly the direction that it takes.

This analysis begins with a thorough case study of the most complex project observed being created, namely the Config-it front end being developed at Talia. This case was chosen because, as the most difficult project, it had the most potential for revealing varied strategies for creating software. The analysis was used to identify the activity systems in which the designer and developer were directly engaged. This revealed how the work of creating the Config-it software was structured.

In the process of revealing this structure, this analysis emphasizes the *aims* of the software-creation activity because these are important for determining what software is

⁵ The word subject is used in this dissertation in an activity-theoretical sense.

produced. The analysis focuses on how the designer and developer come to focus on certain aims and not others, and the role of artefacts in determining and achieving aims.

In the second part of this analysis, the software-creation work at all four organizations was studied, although Talia is again featured as a larger case study. The focus of these analyses was to reveal the network of related activity systems around the software-creation activities in these organizations. These analyses also emphasize how other activities influence particular software-creation activities to focus on certain aims. The chapter concludes with statements about how the work of designers and developers is structured and the role of artefacts.

Activities at Talia Established to Create the Config-it Front End

Talia was the largest and most complex organization I studied. Also, their multidisciplinary project team had the most challenging software product because it was one component of a large, complex, real-time system of software and hardware products. All other organizations, projects, products and end-user environments were simpler by comparison. The initial understanding of the designer and developer at Talia was that the product team's clients were large organizations, and the end users of the product were employees of these organizations. However, as the project unfolded, they learned that some end users were employees in small client organizations and some end users were not associated with a client organization, but were interested in managing their own configurations.

The team I observed was transitioning from a waterfall process (a fairly rigid, prescribed, document-centric process) to a more light-weight Agile process called Scrum (Schwaber, 2003). Scrum, like other Agile processes for managing software, supports collaborative, multidisciplinary teams to produce software in carefully determined

incremental releases that reflect their software project's changing priorities. Increments are released in sprints, a phase in the Scrum process. The Config-it project was one of the first projects at Talia to use Agile methods. To plan each increment, a Scrum team holds a day-long workshop to determine what the outcome of each sprint should be. On a daily basis the Scrum team, together with interested stakeholders, holds brief stand-up meetings where team members report on their progress and state their plans. At Talia most members of the Config-it software team were completely unfamiliar with Scrum at the beginning of the project, and only began adopting the practice of stand-up meetings while I was there. I observed part of the first sprint, all of the second sprint, and the beginning of the third.

To develop Config-it, two separate teams developed the front and back ends, but I only followed the front-end/interface development work. The design work to build the interface was extensive and included designing a windowing system and a custom framework for programming dozens of user interface features. The work involved team meetings roughly every week with few team member interactions in between. When I began there was no interface whatsoever, the team was selecting new tools, and switching over to a new process. On my last day of observation, 60 days from the start, tools had been chosen, a windowing system had been designed, a framework for implementing the user interface was coming into existence, and the team was adapting to their new process.

While this analysis is based largely on observations, I used interview data to clarify the motives of the activity and the objectives of the key participants in the activity. I also had several informal, but lengthy, discussions with the senior analyst who launched the product, who described the product to me as one element of a system of products, and who informed me of the original business motives for the project. As well, I had two

meetings with the designers' manager, who explained the role of Talia designers on software projects generally. I also interviewed the designer, the developer, and the leader of the project management team who was also the developers' manager. Thus, this particular analysis, although based primarily on my observations, was informed by five sources occupying six different project roles.

I entered the workplace eight times, on days when the Config-it project team had scheduled a meeting. Between meetings, the team worked at their desks, conferring with colleagues in their discipline or with the leader of the project management team. Because the designers and developers also worked on other projects, I kept in touch with them between team meeting days via phone calls, asking them about their progress and plans for future meetings. In total, I observed eight team meetings and have notes and artefacts relating to four other relevant team meetings that my participants described to me. In 11 of 12 meetings both the designer and developer were present. Designers and developers interacted almost exclusively through scheduled meetings in which other team members were always present. Talia had no impromptu meetings and I never saw the designer and developer chat, although each did chat with other members of their own discipline. The team also had full-day planning workshops every six weeks and daily stand-up meetings to manage the project work, but there was no opportunity to observe these. There was only one workshop during my observation period, and, although I was welcome to attend it, I accidentally did not receive the email invitation. Also, because of security issues, it was easier for my participants if I arrived after their stand-up meetings, as I was moving around the building less this way.

An activity systems analysis is challenging to undertake. It requires finding out how teams order their work, which is particularly hard when the work is creative. Part of

the challenge is the precise identification of the collection of elements that comprise an activity, which I revised many times as my understanding of the work deepened. Even identifying and naming the activities was challenging. For instance, I only became aware of the existence of a separate back-end team in the latter stage of the observation phase, when the designer I was following met with a member of the back-end team to discuss details of their shared interface. This revealed a new activity as well as the complete independence of the front-end and back-end teams. Until that point, I had assumed the front-end team was developing the entire product. Another revelation occurred 30 days into my observations when the front-end team learned they were creating not one front end, but three similar front ends for very different groups of end users and environments. However, to reflect how the designer and developer saw their work, as variations of one front end, I have named their activity *Creating the Config-it Front End*, rather than *Front Ends*.

New activities are created when it is very important to achieve something that requires substantial resources and coordination. A second activity, that I called *Managing the Config-it Front End Project*, involved almost all of the same people as the *Creating the Config-it Front End* activity, but in this activity individuals had different roles and there were different rules, tools, etc. A strong concern for completing the Config-it project was the primary reason for a separately organized team activity to manage the project. Next I describe these two activities.

Creating the Config-it Front End at Talia

The Config-it front end is a user interface for a very complex Web application that was to be a complete rewrite of a graphical user interface (GUI) for an existing product. The interface was a complete change in style (Web applications are more

accessible, but also more restrictive) and a windowing system had to be designed and implemented as part of the project. A multidisciplinary team had been assembled specifically to work on the project.

This description of the activity to create the Config-it front end emphasizes the perspective of the designer and developer and the aims of the activity. The analysis begins with the big picture, the business motive, and the desired outcome of the activity, thereby addressing the reasons for the existence of the activity and the most important product attributes. Then, the object of the activity is described along with more specific objectives for the software as they were described to me by the designer and developer. Collectively, I call the motives, objectives, and outcomes *aims* because all three of these elements served to draw the activity toward its completion. I then describe the way the team had been organized to meet the activity's aims. A short section then addresses contradictions that are present in the activity, particularly those that relate to aims. Then I describe some of the tacit rules that influenced the behaviour of the designer and developer as they attempted to resolve contradictions and achieve the desirable aims of the software.

This description emphasizes aspects of the work that relates to setting and adjusting the direction of the project and the mechanisms used to achieve project aims. Throughout all of the work on Config-it toward achieving software aims, artefacts were used extensively, and I emphasize the key ones in the descriptions below.

The big picture: Motives and desired outcomes. In software-creation work the motive, as understood by those engaged in it, is the overall business reason for the activity. These business motives meet business needs. Motives are the reason for the activity, incite activities, and explain the activity's existence.

The Talia team existed to create the interface for the software product Config-it. Further, the designer and developer both explained in their interviews that this interface was meant to satisfy the product owner who represented the organization's clients. It was also meant to be a more maintainable product because the current version of the product was incurring major costs every time a change was required. Robustness and performance had been the previous coding priorities.

Certain roles within the team and certain artefacts helped to achieve the business motives. To ensure that the product would be created, the product plan, used by the entire team, reflected the state of the user interface throughout.

To satisfy the product owner, who wanted a significantly improved user interface, a designer had been assigned to the project. The designer's role was to create the drawings for a set of windows that would be the foundation for producing a more client-friendly front end. The designer presented drawings of a set of proposed windows in meetings where the developer, product owner, and other team members assessed the "feel" of the proposed interface. Another organizational committee was determining a standard look for the interface icons and elements.

To ensure software maintainability, the developer I followed, known as the framework developer, had been given the task of creating and developing a software framework/platform for the user interface. This framework was to be used by other developers on the team to implement the user interface feature-by-feature. Other developers reviewed his documents, which they called "design documents," but which I call "development documents" to avoid confusion with documents written by designers about the user interface design, which were also called "design documents". The development documents contained the developers' plans for structuring the software. In

the development documents the developer had proposed a structure for the invisible software object that would be the user interface platform. He had been careful to ensure this framework was compatible with the designer's designs.

The assumption in assigning such specific tasks to the designer and framework developer was that the work of these two people would automatically be transformed forward into the future work of the feature developers, who would be building the interface, using the windows designed by the designer and implemented in the developer's framework. This tool would provide significantly more than a set of software objects to draw windows because it would ensure important project motives (a usable and attractive interface and maintainable software) were automatically achieved in the future work of the feature developers.

Other motives for the activity were revealed to me by a senior analyst who had originally launched the project, but who was no longer involved with the project. These motives included completing an important product line and producing a product that was superior to the offering of two major competitors. These motives were not reflected in the interviews with the designer or the developer, nor in the artefacts that the team used—I did not see an examination of competing products, nor did I hear talk about Talia's product line. I concluded these motives may have evolved, that the product manager had addressed them, or they were forgotten, lost, or not transmitted. For whatever reason, the original motive for the activity no longer matched the motives as understood by the designer and developer. (As surprising as this may seem, this happened at other sites too, although space does not permit going into these other cases).

Sharing artefacts allowed the different motives to be balanced against each other. For instance, the project plan could be modified to allow more time for designing the

interface, or the interface could be simplified to meet important deadlines or “implementability” objectives. Both the project plan and the drawings of windows were widely shared and modified by others on the team. However, the developer did not share the development document for the framework widely because he assumed that nondevelopers would find these documents meaningless and would be unable to relate them to their own work. Therefore, nondeveloper team members never saw, and could not modify these artefacts. This meant that discussions about trading off software objectives such as the “feasibility of the software” or “the ease with which it could be implemented” with other project aims would be difficult.

When motives were not reflected in the activity, no one on the team was invested in the motives and no artefacts existed to ensure their achievement. In such cases team members did not focus on the motives and there was no possibility of balancing these motives with others.

The most important desired outcome was a complete and timely rewrite of the previous version of Config-it’s interface, but other subsidiary desired outcomes aimed to ease maintenance work and future product development. These secondary outcomes had the potential to come into conflict with the primary outcome. For example, the Config-it front-end team planned to use two emerging organizational standards. One aimed at standardizing the look of all Talia software products and the other aimed at standardizing the tools used to develop Web applications. Separate activities (as evidenced by the existence of separate teams) had been established to develop these standards and their work was not yet complete, resulting in some tensions on the Config-it front-end team. Because of delays in the standardizing work, the team had to select their own development tool set and the designer had to design windows with placeholder icons and

colour sets. The developer felt the choice of a development tool set was very important, while the designer felt the new “look and feel” of the product was very important.

Another desired outcome for the team was garnering experience with Agile methods and, to this end, the team had organized themselves to manage the software-creation process using Scrum—and were one of the first software teams to use Agile methods within Talia. The team was learning about Scrum through workshops, but this was taking time and disrupting their work and resulting in delays as they reorganized their work. The designer and developer were variously invested in these outcomes. Both were adjusting to the new practices required by Scrum.

They both found stand-up meetings to be problematic. The developer found them a waste of time because the nondevelopers on the team could not help him with his problems when he was “blocked,” and estimating the amount of time to complete the framework was proving very challenging because this work was nonroutine. The designer had problems committing to the stand-up meetings because of other commitments he had to scores of other projects (although numerous, many were small commitments like the redesign of a single icon).

Subsidiary project outcomes were interdependent, so artefacts were used to help the team set priorities here, which was not easily accomplished. For example, in one of the first team meetings, the developer wanted to discuss the merits of three different tool sets. However, he could not show the designer the tool sets or discuss technical pros and cons, so he presented a series of applications developed with the three different tool sets to illustrate their varied capabilities. He was excited about working with these different technologies. By demonstrating Web applications to the designer the developer was indirectly posing the question, “Could you see yourself successfully designing the

Config-it front end if I used this tool set, or this one, or that one?” The designer, unfortunately, misunderstood and thought the developer was pushing him to add all sorts of “bells and whistles” to his designs because the developer’s demonstrations included so many different possibilities. It was several weeks before a decision was made about the developer’s tool set. By then, the developer had tried implementing a minor feature using each tool set and in the process learned that not every tool set would allow him to create the standardized look Talia wanted. Further, all toolsets limited the potential feel of the application in different ways. These considerations overrode any preferences he had for the tools.

Similarly, the designer, whose work was to create the product’s windowing mechanism (its general look and feel), presented a first mock-up of the product to the manager of the developers. Together they discarded certain windowing techniques such as wizards, floating windows, and stacked windows, but agreed that two ideas that would need specially-designed windows were worth pursuing with the team. These were “object wells” and “filter shortcuts.” To get wider approval for these ideas, the designer presented them in the form of a second mock-up in the next team meeting, but at the same time he also presented a third mock-up that depicted a very simple way of managing windows.

The team, especially the developers who were experiencing some anxiety about implementing anything “too tricky” for a Web product, preferred the third mock-up. This revealed that, for the developers, “implementability” on the Web took precedence over “ease of use”. This view was also shared by the rest of the team because producing the software was a high priority. The idea of object wells and filter shortcuts was dropped.

Unfortunately for the entire team, the mechanisms for discussing a design proposal's implementability were weak, and essentially came down to the informed opinion of the framework developer or the technical lead, who were both new to Web development. Several meetings after the meeting where the idea of service objects and filters was presented, the designer presented an idea about end users being able to interact with "actionable items in lists" using another mock-up, which the team also did not endorse.

Despite all of the rejections, the designer's presentations did help the developer see the direction the designer wanted to take with the windows he was designing and they helped the designer understand what was and was not feasible. Shortly thereafter, the developer selected a tool set. By the time the designer presented his 7th prototype, he had mastered a part of the tool set the developer had selected. This part allowed him, for the first time, to create the interface himself using "drag and drop" methods, and to automatically produce the code for the visual aspects of the interface independently of the developer. This transformed designer-developer interactions in a way that was seen as a good thing from both the developer and the designer's perspective. Thus, the designer's mock-ups helped the developer choose a tool set, which in turn changed the designer's way of working and made designer-developer interactions easier.

At subsequent meetings the designer continued to present his ideas. At one meeting he had a fellow designer present his work for a non-Web project on simple and complex filtering. However, by then, the developer was writing releasable code using his newly acquired tool set, and was providing what the team called "serious pushback", meaning challenges to the designer's ideas. The developer was beginning to get a real idea about the magnitude of the work ahead, even for the simplest of the designer's

proposals. The designer, he said, had “too many good ideas” and “we can’t do them all.” The developer, who could not say, “Look at the corresponding size of the work you’re proposing for me with this idea you’re proposing,” because software is invisible, provided various forms of resistance instead, such as criticism, reluctance, mild expressions of incredulity, a lack of enthusiasm, and generalized complaints about workload.

The designer and developer could not work independently of one another on issues to do with project outcomes and used artefacts to balance competing priorities. On the whole, using artefacts was easier for the designer. The developer experienced difficulties communicating through artefacts and sometimes resorted to simply resisting the designer’s ideas. The artefacts available to the developer were not understandable to the designer, and the developer had no means of representing the time, effort, complexity, and risk associated with implementing the designer’s ideas.

To summarize, the presence of the designer and framework developer on the team were directly related to the motives for the activity. The designer and developer were aligned on the motives for the project, but not with the senior analyst who had launched the project, which raised a question about whether or not the real business needs of the organization would be met. As a direct consequence of the project motives (the need for usable and maintainable software), the designer and developer were responsible for creating artefacts (i.e., sets of windows and a framework), which would ensure future feature developers would readily achieve the project motives. The project existed because of the motivation to produce software. The benefits of other outcomes were mainly for future project work. The designer and developer were variously aligned on the different outcomes, but their work on them was nevertheless interdependent. Whether they were

working to achieve important business motives or subsidiary outcomes of the activity, the designers and developers created and shared artefacts to balance these aspects of the project against one another, although this was easier for the designer than for the developer.

This analysis now looks more closely at the software, that is to say, the object that is being transformed by the team.

The team's object, the Config-it front end. The object of the team's activity was the product they were creating. The team's product was unusual, even in their industry sector, and it was intended for a highly complex environment. Moving from a GUI interface to a Web interface was a large change, particularly because the interface was also meant to substantially increase the usefulness of the product. The prior GUI interface was not designed, and had not been implemented, with usability as a top priority. Objects in the interface, such as service contracts, groups of contracts, and regions were organized into a single giant tree that end users had to navigate to find items. The developer informed me the tree was an exact duplicate of an underlying database that held all the service objects. Although the product was very robust, no one liked the interface. Most notably, the designer informed me, it took time to find items and it was difficult to group similar things together (e.g., all of the service contracts for an individual). The team's review of the current product revealed many problems: it was full of technical jargon, major inconsistencies regarding basic assumptions various developers had made about the business model, and minor inconsistencies such as the same operation producing two different kinds of results in two contexts. The product owner wanted all of these issues resolved. Understandably, the new design was to start from a blank page.

To design from scratch the designer and developer needed to know the nature of the product and they needed to decide on the underlying model for the windows. Such artefacts were necessarily abstract, but as important as the drawings or product plans that were mentioned in the previous section. Abstractions were useful for determining the direction of the product, so therefore had a large impact, and I was interested in how the designer and developer worked with them. Meetings to explicitly determine such abstractions did not occur. Instead, the topic of abstract models only came up from time to time as a side issue when discussing a mock-up. The team determined abstractions only indirectly as illustrated in the following description of the Config-it front end's development, which shows how abstract artefacts are used in team meetings.

In one of the first meetings I observed, the product owner set the direction for the Config-it front-end work. At one point, just as an aside, he stated that Config-it was a “service manager” (i.e., software that manages customer services). He also said he wanted Config-it's front end to be “like XYZ,” another software application at Talia, which had a graphical user interface, rather than a Web interface.

Several weeks later when observing the designer at his desk, with evident excitement, the designer explained to me that he saw the product as a “network monitoring system,” a term that describes all software that collects and displays data on elements in a network. He showed me how the set of windows he was designing mapped to a standard list of network management functions he had found online. This abstract model of the software allowed him to know what set of windows he had to design, and consequently, when his work would be done. He also expressed his frustration with the requirement that the interface style be “like XYZ” because he was unsure what aspects of XYZ's interface he should be emulating and why.

In their next meeting, in the process of responding to another one of the designer's prototypes, the developer proposed another software product, ABC, as a model for a part of the interface. ABC had a flexible, yet practical, windowing model. Adopting this product's treatment of windows as a model would mean the developer would be able to easily understand how the user interface should work. The designer subsequently spent time learning about ABC and adopted aspects of its windowing model in his subsequent designs.

In the last meeting I observed, 60 days from the start of my observation period, the senior developer who was presenting the requirements document for the user interface began by describing the product as "a content management system" (i.e., software that supports the collaborative creation and updating of data). As the team responded to the specific requirements in the senior developer's document, ABC was again mentioned as a model for the user interface style, along with a third product that had not been mentioned before—RST. Without explicitly discussing it, XYZ was no longer being used as a model for the user interface.

The product owner, the designer and the developers were clearly searching to identify the style of the user interface and a model for their unusual product, neither of which they discussed explicitly, so there was no due consideration of the pros and cons of various choices that were being made, and yet these choices had a significant impact on the shape of the software. Individuals appeared to try a style or model to see how it might influence their work. Styles or models were dropped if they proved unhelpful in team interactions.

Because each of the topics came up at about the same time, the topic of interface style and product nature appeared interdependent. This suggests that to determine the

user interface style, and to decide on an appropriate set of windows to be designed, a certain amount of general knowledge about the nature of the product is necessary. That is to say, designers, even designers of a general windowing system, cannot work without product knowledge—knowing something as basic as the type of interface helps. A service manager, for instance, might not need pop-up windows, but a system that responds to alerts and alarms would.

By revisiting these topics in meetings, the designer and the developer, and in fact the wider team, showed they needed agreement on these topics to guide their individual work. In the designer's case I saw him using the information to check for the completeness of the window set he was designing. In the developer's case, his suggestion for a particular windowing style primarily reflected his concerns about complexity. However, his suggestion to use a style he knew well would allow him to make good guesses about how to implement windowing operations such as closing or minimizing a window. This information would be missing from the designer's drawings and might or might not be present in the user-interface requirements document, which the technical lead and product owner were writing, without the designer. Abstract models and exemplars were therefore important artefacts to both practitioners. However, abstract artefacts are evidently challenging artefacts to use because the desirable abstract qualities of the exemplar were ambiguous and the importance of the model, its merits and drawbacks, were never expanded upon.

In summary, the first interface style and the model of the nature of the software were proposed by the product owner in one of the first meetings I observed. These were mentioned prior to the existence of any drawings of the interface, and I understood these as the very earliest prototype of the Config-it front end. Further into the project these

abstractions became hidden and largely undisputed parts of the more tangible mock-ups the designer produced. Such artefacts were not the focus of discussion, and were not debated or discussed in detail, although they were influential in shaping the software.

Having now explained the object of the activity, how abstract artefacts aided in its transformation, the analysis now looks at the issue of the important attributes of the object (i.e., the objectives of the software).

Objectives for the Config-it front-end software. Objectives are closely associated with the object of the activity and capture a desirable social or cultural attribute or quality of the Config-it front-end software, which is the object of the activity. The object and objectives are intertwined because the object, when fully transformed, should ideally embody all of the desirable social and cultural objectives. Objectives are not the primary business reasons for the activity, but are nevertheless important qualities that the team hopes will be a part of the product. Objectives, unlike motives which incite an activity, serve to clarify the most important outcome of an activity. Example objectives for software include usability, maintainability, or robustness.

Config-it's objectives varied depending on the team member asked. For instance, when interviewed, the designer expressed concern for the product's maintainability, on-time delivery, responsiveness, and testability; whereas the developer expressed concern about the Config-it front end being well-designed (i.e., maintainable), modern, and installable.

Differences on the views of the objectives of the activity were a source of positive and negative tension on the project—positive because work on objectives focused an individual's activity and ensured the product would have the corresponding qualities, but negative because objectives had to be balanced and team members were invested in

certain qualities more than others. Talk about objectives came up regularly, but indirectly, in conversations about the designer's mock-ups, which were generally the focus of meetings. These mock-ups were something that everyone in the team could relate to because everyone could easily understand the visual language used to generate the drawings of windows. The drawings were the artefact that most visually resembled the team's object, and everyone could relate their responsibilities to the team's object.

Concerned stakeholders outside of the activity simply assumed some objectives would be achieved in the completed software. "Robustness," for example, would be ensured by a quality-assurance team, while the "timely creation" of the Config-it front end was a major concern of the Managing the Config-it Front End Project activity that carefully tracked the project's progress. Such objectives will be discussed in detail in a subsequent section.

Other objectives were so important they were the motive for the activity. For example, "maintainability," a desirable objective on many projects, was tied to the business reasons for the Config-it project.

Objectives that were not assigned to a team member were a source of tension for the team. For example, all agreed responsiveness was an important objective, and specific minimum response times were suggested in the requirements document. The senior developer and product owner, who were the authors of the requirements document, believed that responsiveness would be experienced through the user interface, and should therefore be owned by the framework developer on the Config-it front-end team because his work was the foundation for the other three feature developers. However, the framework developer deflected responsibility for it, arguing that responsiveness was the

responsibility of the feature developers and/or the developers working on the back end of the application.

Some important objectives emerged at surprising moments. For example, 60 days into my observation period, in a requirements review meeting, the designer was describing a scenario to the developer in which a user logs in and the organization's logo/banner is displayed in a specially designed window the designer was working on to display Talia's logo. The product owner intervened into this discussion pointing out the login feature needed to be "customizable," rather than fixed, so that it could display a variety of banners and not Talia's logo. He also pointed out that the login feature needed to be very "secure" because of its varied contexts of use. This introduced two new product objectives—"customizability" and "high security."

The team missed these important objectives, he said, because there was a "recurring problem or misunderstanding in the team." He then clarified that there were actually "three distinct groups" of users of Config-it and therefore, in his view, "three user interfaces to be designed." He named and described the three user groups, identifying one of the interfaces as "a service manager" and the other two as "secure portals"—one of these for internal staff and the other for external users. He then explained that the Config-it front end, which was really one front end from the Config-it front-end team's perspective, was meant to look like three different products from the clients' perspective. He also explained how these three versions of Config-it fit into their clients' business.

By failing to provide this information at the outset, the product manager had assumed the designer and developer did not need detailed information about the product in use to fulfill their roles. Although the designer and developer were working at a level

of abstraction that meant they were not creating the user interface directly, but were building a set of windows and a software structure for the feature developers who would build it, they still needed to talk about concrete and dynamic aspects of the future product in use. This example shows how scenarios helped them fulfill this need to work out details while also allowing serendipitous information to be revealed, albeit belatedly.

To summarize, just like motives, objectives served a role in directing the work toward something and, in absentia, away from something else. This review of objectives showed there were many ways to bring objectives to the team's attention and to ensure that some objectives were given higher priority than others. One important, but not well-recognized method for revealing objectives was through scenarios that provided information about assumptions team members' made about the product in use.

Having reviewed the motive, desired outcomes, product, and product objectives, I now describe how the work was divided among the team members to address these motives and achieve product objectives and outcomes. Many team members have been mentioned already, but this next section provides a comprehensive description.

Division of labour within the Config-it front-end team to address project aims. A complex software product such as the Config-it front end is developed by subdividing the labour. In this section a connection is made between the division of labour among the team and the product objectives that concern each team member. An organizational chart would not reveal this division of labour because this team was assembled only for the duration of this one project, reported to different managers, and only shared a common manager about five levels up the organizational hierarchy. Usually, people's job titles just had to be expanded a bit to describe their role on the Config-it front-end project (i.e., their responsibilities toward the team's object). In the few instances where a job title did

not reflect the person's role on the team, I created a phrase to describe the person's responsibilities vis-à-vis the team's object.

Eleven people were involved in creating the Config-it front end. The developer was the user interface framework architect. He was responsible for writing the code for a framework that would be used to create the front end. This was the first time he was developing a significant Web application, and the first time using a Web development tool set, making him inexperienced for the specific project, despite many years of experience as a developer on graphical user interface projects. He was concerned that the application be maintainable, modern, installable, "setupable" and "upgradable."

The designer's role was to design a set of suitable windows by graphically specifying their appearance. Although the designer normally designed individual graphical elements, such as icons, he was eager to design windows. An industrial designer by training, the designer had followed a series of courses in user-centered design offered by a reputable organization. The designer was concerned that the product would be responsive, on time, a superior platform for future developers, testable, and that it satisfy existing customer complaints regarding usability and usefulness.

The product owner, also known as the product line manager of Config-it, determined the features and liaised with the organization's clients. He was responsible for the software meeting the business needs by ensuring it was marketable and maintainable.

Three Config-it front-end feature developers were expected to eventually develop the user interface feature-by-feature using the work of the framework architect, which would incorporate the work of the designer. These developers would make the connections with the back end. They were concerned about correctness and completeness.

A senior developer was the technical lead. He worked separately with the product owner to write the requirements specification and to provide technical advice to the team, although Web-design was not his specialty. He was concerned about the feasibility of the project plans and the “completeness” of the project.

The developers’ manager, who was the team coordinator, pulled together resources to help the team proceed on their work for the Config-it front end by setting up team meetings, meetings with other people, and by resolving conflicts. She was concerned that the project was sensible, “completable,” and acceptable to the product owner.

The designer’s manager was the person responsible for broader design issues at Talia. She was a key member on a committee designing Talia’s new look. Although present at most meetings, she only became engaged in the project when design issues were very broad, such as the impact of the tool selection on the look of the product or the possible use of a general filtering mechanism designed by her design group. Her objective was inter-product consistency.

The tester and documentation writer were not very influential and were mainly concerned about understanding the product and its impact on the complexity of their future work. They were concerned with simplicity, usability, and also that the Config-it front end be consistent with other Talia software.

In summary, these 11 individuals, with differing responsibilities and overlapping and competing objectives, were assembled to create the Config-it front end.

Team rules and their impact. Various tacit rules controlled the software-creation activity. Some of these rules originated from the organization’s hierarchical culture. For instance, one tacit rule was that the product owner, who represented business issues, had

the final say regarding what the product would or would not be. Designer-developer interactions were heavily influenced by the opinions of the product owner and the requirements document the product owner produced with the help of the technical lead. The designer and developer, along with everyone else on the team, were always hoping to have a quick word with the product owner after meetings broke up. The product owner's influence was a carefully negotiated aspect of team interactions because interactions with the product owner determined the relative importance of the activity's motives, objectives and desired outcomes.

However, the product owner took care not to be overly influential. He would often remain silent for some time in meetings before asking a question or speaking up, seemingly allowing all of the issues to be aired. His requirements document also solidified discussions in previous meetings, rather than being dogmatic views of what the product should be. The mediating presence of the product owner and the tacit rule that his wishes were to be followed ensured that the overall motives of the activity had greater weight because the product owner's responsibility was the achievement of the business motives. At the same time, because motives and objectives were interrelated, he also wanted to make sure that other objectives and long-term outcomes were properly balanced off against each other, and against the project motives.

Other rules originated from the techno-centric nature of the organization. The techno-centric nature of the organization also had an impact on designer-developer interactions. Because knowledge about technology was highly valued at Talia, the designer's more artistic work was generally less well-understood relative to the developer's work. One indicator was the assumption that the designer did not need much concrete knowledge of the application being designed. Not only was there little

discussion about end users, the marketplace, or context of use, but designers had virtually no contact with clients and end users (although to be fair, this was true of almost all Talia employees, save the sales staff). This presented a serious impediment to their work, particularly for projects like Config-it that demanded much more of designers than their skills as graphic artists.

Another indicator that design was not well understood at Talia was that the feature developers, and not the designer, were responsible for the flow of end-user interactions, even though the designer was trained in this area. Also, the designers were referred to as “industrial designers” or “graphic designers” by the developers and other team members, with the result that the designers’ training in user needs analysis, usability testing, or interaction design were not well recognized or utilized.

Talia’s hierarchical and techno-centric culture, seen as pervasive rules here, had an important impact on designer-developer interactions, particularly in determining the ability of the designer to act.

Tools for achieving motives, objectives and outcomes. The team members used various software tools to create images, write documents, code and, occasionally, model the software. However, the designer and developer were generally unaware of each other’s tools, except when changing tools had an impact on the other’s work. The example of the developer selecting his tool set, which was previously described, and how this act changed how the designer and developer worked with each other, illustrates this.

However, some tools were shared. Meetings, as tools for raising joint concerns, were one of these. The team met every week or two. Meetings were used to investigate potential development environments, to clarify what should not be built, to discuss concrete suggestions about what to build, to understand the product in use, to explore the

relevance of ideas being considered in other products, and to agree to a contract about what to build. Meeting-room management software and shared calendars helped to arrange these meetings.

The meeting rooms were a well-used aspect of the physical workspace. However, other barriers prevented other uses of the physical workplace for shared work. For example, proximity to other team members was an issue. The entire team was distributed physically across various floors of the same building, although developers were co-located and designers were also co-located. As such, the designer and developer mostly worked at their desks in distant cubicles; they had work-related chats with nearby colleagues and their managers, but never, to my knowledge, with each other.

Crowded cubicles also prohibited chats and impromptu meetings, and nearly prevented me from entering the workplace at all—the designer’s manager initially claimed there was not enough room for me to conduct my research. When I arrived, before I could enter the designer’s crowded cubicle, a chair was borrowed from elsewhere, and the removable arm was taken off the chair so that the chair could fit in the cubicle, allowing me to sit and watch the designer at work. Also, a condition of my entry was that I not disturb the other designers in adjoining cubicles.

The developer’s cubicle was larger, but hardly spacious. The physical barriers in this workplace therefore dissuaded face-to-face, two-person chats or impromptu meetings and favoured larger more formal scheduled meetings in general-purpose boardrooms. The opportunities for making meetings happen were important because these were times when important disparate motives, objectives, and outcomes were balanced off one against the other. Also, Scrum emphasizes the importance of team members being co-located ideally in team rooms.

Designer-developer interactions were therefore mediated by the selection of new tools, meetings, and the physical factors of the workplace, as well as the artefacts that team members brought to meetings, such as the designer's drawings and the developer's demonstration applications.

Resolving tensions between motives, objectives and outcomes. Many tensions or contradictions were present in this activity. The ones discussed here related to the activity's motives, desired outcomes, and product objectives.

One tension in the project was the conflicting vision of the product among the team members. These visions embodied a collection of different motives and objectives and varying commitments to outcomes. The developer envisioned a product that was *modern and implementable*. The product owner had a strong vision of what he did not want (perhaps as a result of numerous customer complaints), but was less clear about what he did want. The designer wanted certain ideas implemented in the user interface (he called it his wish list), but more than this he wanted to align all parties to one vision— particularly the developer and the product owner, whom he saw as being on “different pages.” The designer did the bulk of the work realigning disparate visions by preparing and presenting two or three alternative drawings of the envisioned windows at almost every meeting. These were intended to be challenging, and they did incite vigorous debates in team meetings.

There were also tensions due to team members' varied understandings of client and end-user needs and the context in which the application would be used. These misunderstandings caused a certain amount of stress and resurfaced periodically in the meetings. Such tensions were difficult to resolve without direct access to the client or end users. Discussions at meetings helped to bring greater understanding. As well as these

discussions, the designer worked hard to increase his knowledge of the previous version of Config-it, its users, and context of use by reading every document he could find. This amounted to an impressive 6 cm pile, including half a dozen white papers, various end-user manuals, usability reports, diagrams of systems, and the odd document written by clients. He made notes, created beautiful images of interrelated service elements using drawing packages (I saw a similar drawing he had created for another project posted in many cubicles), and he also “played with” the previous version of the Config-it software. The latter he did by running a test version of the previous version of Config-it that various people used at Talia, which was seeded with sample “toy” data.

Some contradictions related to the activity’s outcomes. One of these was the tension created by transitioning between their old management process and Scrum. For instance, although their new practice emphasized the equality of team members, the product manager, the designer’s manager and the developer’s manager still had more power. Also, although Scrum advocates cutting back on documentation writing, there were issues over how much documentation to cut.

There were many tensions surrounding the project. This section emphasized tensions relating to motives, objectives, and outcomes, which were not independent.

Summary. The work of the designer and developer on the Config-it front-end team at Talia was primarily to create a novel user interface that would satisfy client needs and be maintainable. The motives of the activity were reflected in the specific roles of the designer and developer and these specialists created artefacts to make progress toward achieving the activity’s motives. Many artefacts were shared to allow checks and balances to be made within the team, although sharing artefacts was particularly challenging for the developer because software is invisible.

The primary desired outcome of the activity was to create the software, but additional outcomes directed to improving the activity itself were also important. Designers and developers were variously invested in different outcomes, and one would think work with these outcomes could proceed independently, but in many cases work on one outcome impacted another. Therefore work toward these outcomes, though ostensibly independent, was joint work, and the artefacts used in designer-developer interactions supported the coordination of work that proceeded independently in general.

The object of the activity was not obvious and emerged as the activity progressed, especially through the use of drawings, abstract exemplars, and models of the software. Exemplars and models were the very earliest forms of existence of the software object and served an important role in that they pulled the activity toward one set of objectives or another. They had an ongoing role as the abstract part of the designer's drawings.

This analysis highlighted the place of objectives and motives (seen as elevated objectives) in the work of software creation. While some objectives were present from the very beginning of the activity, others were revealed when reviewing concrete aspects of the product. Objectives were shown to influence the direction of the work, and the team showed by their actions that they were most comfortable when objectives were "owned" by someone, rather than being shared by the entire team. Objectives needed to be balanced against each other, and therefore progress toward objectives needed to be shared. The analysis also revealed a strong link between objectives and the object of the activity in that a deepening understanding of the object revealed further objectives. The study showed that artefacts can serendipitously serve the role of bringing unrecognized objectives to the fore. This was the case with the revelation of the requirements for a customizable and secure login feature that emerged during the course of a discussion of a

login scenario that was initiated because of the presence of the designer's drawings of the login screens.

Table 12 lists the rough priorities among the motives (which are associated with the activity), objectives (associated with the object), and outcomes that in a sense competed for the attention of the designer and developer. The priorities are listed from most important to least important as demonstrated by the team actions. Table 12 clearly shows how higher-priority aims had activities created to achieve them, were assigned to a specific individual, or were tracked by artefacts. In some cases, such as for the product owner, these aims were tied to performance reviews. Lower priority aims were not attended to, were often difficult to measure, were not a motivation for the project, did not have a person responsible for ensuring the aim was achieved, and were not tracked or were completely "off the radar."

Table 12 *Some Aims for the Config-it Project by Perceived Priority*

Priority	Type of aim	Specific aim	Team resources to achieve the aim
1	Motive	The software needed to be created	This motive was the reason for the Creating Config-it Front End (and Back End) activity's existence.
2	Motive	The software needed to be on time	This motive was very important and would also impact the product owner's performance review. Artefacts existed to track this.
3	Motive	Increased maintainability	This motive was so important a special role was created for the developer. He was required to build a framework to ensure the software's maintainability. A project plan tracked progress toward achievement of this motive.
4	Objective	Testability	This objective was owned by a single team member, the designer.
5	Objective	Customizability, security	These objectives were discovered serendipitously during the activity and were diffusely owned by various team members.
6	Objective	"Installability" and "upgradability"	These objectives could not be measured easily and were easily forgotten. These were minor concerns of the framework developer.
7	Outcome	Trialling Scrum, a new management process	These outcomes may improve the activity and future activities. The Scrum master was invested in achieving this outcome.
8	Objective	Responsiveness	This objective was not owned by a team member.
9	Objective	Competitiveness	This was a forgotten, missing or miscommunicated objective and no resources were assigned to it.

Designers and developers worked to achieve competing aims. At several points throughout the description of the activity, there was a tension between how much knowledge of the concrete activities of clients and end users the designer and developer required to proceed with their work. At several points the designer and developer indicated they needed more information than the product owner assumed they needed.

The analysis showed how tools aided the team to meet objectives and that having a meeting was an important shared tool. Because meetings were tools used by the group to advance the work, I saw meetings as important artefacts mediating designer-developer interactions, even though meetings have a significantly different quality in comparison to more conventional artefacts such as drawings, lists, or models.

Although many purposes for meetings exist, this analysis showed that key decisions that advanced the work happened in meetings. In particular, meetings provided an opportunity for clarifying what direction should not be taken, proposing a new direction, responding to a direction that had been taken, committing to a direction, or deciding how to proceed. Meetings were enabled, or not, by the physical environment and the tools for setting up meetings.

An analysis of the tensions in the activity identified important contradictions relating to conflicting motives, objectives and outcomes. This section revealed the structure of the activity involving the 11 specialists at Talia and the place of designer-developer, artefact-mediated interactions within this structure. Figure 2 provides the traditional depiction of this analysis.

Organizational Needs
Increased market share
Balanced budgets

Organizational Motive
Production of Config-it 5.0 interface,
Satisfaction of client requests,
Reduced future maintenance costs

Desired Organizational Outcomes
Complete rewrite of the user interface of
Config-it 4.0
A new usable development platform for
Config-it 5.0
Application of a new 'look' for Talia
software products

Articulated Project contradictions: Conflicting visions of the project; The pull between innovation vs. simplicity; Multiple understandings of client needs and end user experience; The tension between methods used in previous projects vs. methods required for the new project.

Figure 2. Creating the Config-it Front End activity.

Managing the Config-it Front End Project at Talia

At Talia, designers and developers also interacted to manage their project. The motive associated with Managing the Config-it Front End Project was to ensure that the Config-it front end was produced in a timely and cost-effective manner. This meant that it was this team's task to work out priorities, to organize the work in a sensible manner, and to track the work. Because this work does focus on the direction and priorities for the project, it was regrettable that I was not able to observe at a workshop or to attend stand-up meetings. My knowledge of this activity is therefore secondhand.

A host of individuals were part of the management team, largely the same people as in the Creating the Config-it Front End team, but with the addition of several new people, such as higher-level managers. The team was following Scrum practices. The developers' manager, whom I interviewed about software processes, took on the facilitator's role as "Scrum master," although usually managers do not take on this role. The usual way to specify the roles in Scrum is to distinguish the product owner, the Scrum master, and "the rest of team." However, a common practice, which was adopted at Talia, is also to distinguish "pigs" and "chickens." The developer was considered to be a pig, which is a characterization of the role of individuals on the team who actually do the work in some understandings of Scrum. If you were not a pig you were a chicken because you are someone who has something to gain by the pigs performing their tasks. A joke about pigs and chickens opening a restaurant and serving ham and eggs was circulating in the team. There was some discussion as to whether or not the designer was a pig or a chicken, the issue being whether or not his "bacon was on the line" in the same way that the developer's bacon was. The team obviously enjoyed this terminology, even

though many proponents of Scrum object to the characterization of team members as pigs and chickens, finding them unhelpful.

The team was transitioning from a “waterfall” approach to a new way of managing the project and residual aspects of the old method were still present in new practices. For instance, atypically, the developer’s manager played the role of Scrum master. Normally, the Scrum master is not also a manager, but the developer’s manager was the only person with some training in Scrum and she was also a skilled facilitator. This meant that the hierarchical arrangement of team members typically found in teams following a waterfall process was maintained and not altered in the transition.

The shared object that the team transformed was the project plan, which was called the sprint backlog in Scrum. The designer and developer were both involved in determining what items would be added to the next sprint, a sprint being a six-week period of work, at the end of which some tangible goal is achieved.

The team’s objective was to produce their product on time, on budget, and with minimum risk. They wanted to reduce future maintenance costs and also reduce customer complaints.

The desired outcome of the project team was a complete and timely delivery of their software product and the creation of a software framework. They also wanted to explore and evaluate Scrum. Both the designer and the developer were invested in these outcomes to varying degrees, and were slightly in conflict with them. For instance, when asked when the project would be done, both said they expected their involvement with the product would “never end.” In their minds, an outcome of a completed software product was an oxymoron. This view stemmed from their previous experiences on maintenance teams where released products underwent continuous further evolution.

Tools used in the work were standard management tools such as Microsoft Excel, which was used to maintain lists. Document repositories were used to share documents. There was one repository per sprint. Typical items held here would be mock-ups, memos, templates for Web pages, and code. Both the designers and developers contributed items.

Several objects were transformed in the Managing the Config-it Front End Project activity. One was the product owner's product backlog, which was a list of changes required to the product. Items were removed from this backlog and broken down into items and addressed in a sprint. Another was the sprint backlog, which was both a list of undone items for the sprint and the team's *burn down chart*, which was a list of completed items. A final object was capacity lists that served to record the amount of time each team member had available to dedicate to the project on a day-to-day basis. This list was maintained by the Scrum master and implemented in Microsoft Excel spreadsheets.

Various rules, mostly adopted directly from the Scrum methodology, controlled this activity. The team held full-day workshops called "planning meetings" every six weeks to break down their work into actionable items that could be added to a sprint backlog. There were daily stand-up meetings to track the progress of a sprint. The Scrum master kept the burn down chart up to date and shared it with the others via a shared document repository. The designer and developer participated in these activities, but with little enthusiasm because of their general resistance to Scrum, which is explained next.

The Scrum team experienced many tensions/contradictions. These all involved setting up the project to succeed. One tension was the dated knowledge and skills of the very experienced developers versus the requirement that the product be "modern," in the sense that it would use modern elements in the design of the user interface. Neither the

developers nor the designers, though both very experienced, had the right skills for Web development. However, the framework developer tried to reduce this risk by working hard to transform himself into a knowledgeable Web designer. He did this by browsing Web sites in search of useful information and exemplars, a tactic he used for many problems, and by helping the designer understand the possibilities and limitations of the new platform by arranging a meeting to expose the designer to similar Web-based applications.

Another tension was the requirement to use Scrum versus the lack of experience with this method. The framework developer I observed was uncomfortable with Scrum. He found that with the novel nature of framework design work, he could not break down his work into tasks with predictable lengths. He was therefore at odds with the use of a *burn down* chart. The designer was also leery of the process because he expected to be heavily involved in the early stages of the Config-it Front End Project and then “on call” until just before the release of the product, when he would become more heavily involved again. However, he believed the Scrum process assumed his continuous involvement.

A final contradiction was the expectation by Scrum that the team be co-located and the reality of the team being dispersed throughout the building. Scrum team rooms, where project teams are co-located, are an interesting purposefully-designed artefact of many Scrum work environments. Because designers and developers worked on multiple projects simultaneously, it was not feasible to group them by project, so they were grouped by disciplines.

Summary. Unlike the Creating the Config-it Front End activity, where the designer and developer were at the heart of the activity, in the Managing the Config-it Front End Project activity the designer and developer were sceptical participants. In

Scrum, artefacts, such as the sprint backlog, the burn down chart, and stand-up meetings, serve to order, track, and set the rhythm of the work. The team members, including the designer and developer, put items on the sprint backlog, completed them, and then saw them transferred to the burn down chart. These artefacts were realized through the use of Microsoft Excel spreadsheets at Talia. The management team attended “stand-up” meetings daily to report on their progress and sometimes members there were then able to help others whose work was blocked. They attended a sprint planning workshop every six weeks when the sprint backlog was again loaded with many to-do items, each of which could be completed in a day or so.

Figure 3 summarizes all of the details involved in the activity Managing the Config-it Front End Project using the well-known activity systems triangle.

Figure 3. Managing the Config-it Front End Project activity.

Summary of Activity Systems Analysis at Talia

Software design and development can be a highly complex activity and certainly one where artefacts are very important. The work of the designers and developers is directed by the motive for the activity, as well as numerous competing objectives and desired outcomes. This analysis has emphasized the artefacts that serve to support the work of the activity itself, such as drawings and demonstrations of software capabilities. The analysis showed that aims were balanced against each other and that resources were applied to achieving aims. The resources applied to achieving an aim were indicative of its priority. The higher the priority, the more attention and resources the aim received.

One very high priority aim was that the Config-it front end be completed on time and this was a motive for a separate activity comprising all the members of the Creating the Config-it Front-end team plus a few others. Because the designer's and developer's interactions were sometimes directed toward this activity, it was also analyzed in some detail.

This case study exposed real software-creation activity in its very early stages, where the greatest uncertainty exists. This project was so new when I began my observations that I was able to see major movements toward establishing a process and defining the object of the activity. It was an unusual project in that an entire user interface was being designed from scratch. I saw this team organize itself by selecting tools for its work, take chances on new management processes, and adopt an emerging standard for the look of Talia products. This was software creation in its most raw and most uncertain moments.

Some of what was written here may have sounded like software creation as it is not supposed to happen, but much of what I observed here echoed events in the other

three organizations where I had already observed, but which I will be unable to describe due to lack of space. Those engaged in the activity did not know the motivation for the activity. Teams learned surprising facts late and explorations about one issue revealed something else about another issue. People misunderstood and did work that was not necessary. Software-creation work is hard. This analysis has revealed in part why this is the case, some pressures on designers and developers, and how this heavily, artefact-mediated work proceeds regardless of inherent challenges.

Interrelated Activity Systems at Jaba, Talia, Quadrow and Telespace

This section describes the software-creation activity and related activities at four organizations, showing how other activities influence software-creation activities. To begin I briefly describe work on software projects at the four organizations I observed to provide an overview of the diversity of the activities across organizations.

A summary of the observational data across the four organizations is presented in Table 13. These data show, for instance, that designers are usually involved in more simultaneous projects than developers. The exception to this generalization occurred at Jaba where the back-end work of developing games on a solid gaming platform was significantly less involved than designing the front-end of the games, which was accomplished by three collaborating designers and the developer. For software work, most organizations had established multidisciplinary teams with multiple developers and a single designer.

Most of the software projects I studied were using Agile or Agile-like techniques, following the trend in software-management processes. These techniques impacted the way software was created and managed. With regard to the management of software there was significant variability, as summarized in Table 13. For instance, at Talia one

management team managed one software project, as explained in the previous section. However, at Jaba and Quadrow multiple projects were managed by a joint project management team. And, in a different arrangement, at Telespace, management of the project I observed was split along discipline lines, (i.e., separate management for the designers and developers loosely connected by a project manager who met only with the higher level managers, and whom I never saw, even though she was in her office at times when I was there).

Table 13 *Product Creation and Project Management Attributes Compared*

	Talia	Jaba	Telespace	Quadrow	Total
Product creation					
Team size	11	7	37	4	59
Designers	2	6	7	1	16
Developers	5	1	30	3	39
Project	Web interface	Game	Touch screen application	Web application	
Number of simultaneous projects					
Designer	many	Approx. 3	many	5-8	
Developer	1	Approx. 5	many	1	
Project management					
Method	Scrum	Agile-like	Classical	XP	3 agile, 1 classical
Leadership	1 Scrum master	1 project manager	1 project manager	1 project manager	
Process	11 team members manage one project coordinated by the Scrum master	Periodic company-wide stand-up meetings about all projects	Separate designer and developer status review meetings for the one project	6 developers, 1 designer plus manager in stand-up meetings review 25 projects	

At each of these organizations, the activity for creating the software was surrounded by a multiplicity of other related activities, one of them being some type of project management activity. These other activities had a variety of relationships with the activity for creating the software. Four examples of interrelated activity systems

surrounding the software creation activity are provided in this analysis to substantiate this claim, beginning with the activity at Talia for creating the Config-it front end.

At Talia some activities existed to directly influence the creating activity, as was the case with Managing the Config-it Front End. Other activities coordinated their activity with the Creating the Config-it Front End activity, such as the activity to develop the back end of the software. Still other activities depended on the outcomes of the create activity, such as the client activities. Most activities had a multiplicity of relationships with the create activity.

In this analysis, which exposes these other activities, I show how other activities influenced the Creating the Config-it Front End team. In particular, this analysis shows how it came to be that the designer and developer focused on certain software aims over others.

I learned about other activities indirectly through the talk of the designer and developer, through encounters with people associated with the projects during my observations, and through interviews of software team members. I was exposed to a variety of activities at each organization. For instance, at Talia, most other activities I became aware of were fully structured, but some were only partially structured. One such activity was the Planning Config-it activity, which was the responsibility of the product owner and the technical lead. Other activities obviously existed, but I had no information about their structure. An example of this was the group of activities I have called Client Activities to Influence Config-it. In these activities, clients attempted to influence the Config-it Front-End team because client organizations had strong views about the software qualities of the next version of the software they would like to receive from Talia. Still other activities were enacted by single individuals, as was the case for the

Documentation Writing activity. Many documentation writers want software to be more “describable,” and this was likely the case at Talia as well. Other activities were conducted by separate teams, such as the Testing Config-it activity, which was conducted by the quality-assurance team. This team was concerned that Config-it be “testable” and “robust.” One activity that overlapped significantly with the Creating Config-it team’s activity, both in time and in terms of its members, was the managing activity, which was concerned that Config-it be produced in a timely manner. All of these activities existed for specific reasons and had their own objects, but the point being emphasized here is that they all had strong views on the software’s objectives. To illustrate just how many other activities there might be, the other activities at Talia and their business motives are briefly described in Table 14.

Table 14 *Activities at Talia*

Activity name	Description of activity	Motive of the activity
Client activities to influence Config-it	Influencing Config-it to be more valuable to client organizations. This is an ongoing activity for Talia's clients. This activity begins prior to the initiation of the project and continues throughout the entire lifespan of the software.	Providing feedback to shape Config-it to meet the client organization's needs.
Strategizing/ analyzing	Determining that Config-it was needed in the marketplace and that it was worthwhile launching a new project. This discussion happened among Talia executives.	Understanding what products are needed in the market.
Planning Config-it	Determining the specific product to be developed. This activity was shared between the product owner and a senior developer.	Producing the right product with limited resources.
Managing the Config-it Front End Project	Determining and tracking specific outcomes and resources for the Config-it project. Creating a plan to finish. This team was largely the same group as the team for creating the Config-it front end.	Ensuring that Config-it is produced in a timely manner, and making specific plans for the use of resources.
Creating the Config-it Front End	Determining the user interface and developing the software for it.	Creating the Config-it interface, satisfying client requests, reducing future maintenance costs.
Creating the Config-it Back End	Determining and redeveloping the required back-end software.	Implementing the Config-it features.
Documenting Config-it	Understanding Config-it to produce a written description of it for clients and end users. A person from the documentation group was assigned to this work.	Ensuring that Config-it clients and end users can use the software.
Testing Config-it	Understanding and testing Config-it to ensure it meets its requirements and is robust. A quality-assurance team would run tests of the software	Ensuring that the Config-it software is error free and meets its requirements.
Deploying and maintaining Config-it	Deploying Config-it on client sites (a month-long process at every site). Maintaining Config-it at these sites by fixing bugs and supplying updated software. Developers from a maintenance team would do this work.	Keeping the client software running at minimal cost to Talia.
Using Config-it	Using Config-it to configure services for people who have purchased the Client's systems.	Tracking and maintaining client services.

Focus-On Charts

To show how other activities influenced the Creating the Config-it Front End activity, I produced a visualization called a *Focus-On Chart*. An example of this type of chart is shown in Figure 4 and I use this chart of Talia's activities here to explain how to read Focus-On charts. This particular chart was designed to show how these activities influenced the Creating the Config-it Front End team to create software that was more suitable to their needs. In essence, other activities influenced, or were perceived to influence, the Creating the Config-it Front End Project team, and in particular the designer and developer, to focus on certain software objectives—and hence the name for the chart.

In this chart a triangle represents an activity system. The name of the activity is underlined and a dotted line joins the activity name and the triangle representing that activity. An activity system such as Creating the Config-it Front End activity is accomplished by a team of organized people. Most of the other activities represented by triangles are structured team activities as well. Some exceptions to this are triangles that represent individual activities, such as Documenting the Config-it software, and triangles that represent a group of activity systems with a similar purpose, such as the various Client Activities to Influence Config-it that were established in client organizations to influence the Config-it software to be more aligned with their needs (i.e., to be more useful to them). No particular notation distinguishes between activity systems, individual activities, and groups of activity systems, although the activity's name is a clue to identifying groups of activities because all of these have the word “activities” in their name.

The central activity that creates the software, in this case Creating the Config-it Front End, is shaded in grey. The chart has been developed from the perspective of the subjects in the greyed-out activity, particularly the designer and developer.

Inside the triangles are listed the software aims that are important to an activity. For instance, for Client activities that exist to influence the development of the Config-it software, it is important that the Config-it software be useful.

Some triangles are represented with solid lines and some with dashed lines. When the triangle is solid, this means I had direct knowledge of the aim that was important for the given activity. When the triangle is represented by dashed lines, my knowledge of the activity's important aim for the software is indirect. Sometimes this is second-hand, perceived knowledge, such as my knowledge of the Using Config-it activities important aims, which I only know indirectly through chats with the designer, and which he had no direct way of knowing himself.

The chart shows how other activities attempt to influence the attributes (i.e., the objectives) of the central activity's object, which is the Config-it front-end software. Some of these other activities were ongoing, some were initiated prior to the Creating the Config-it Front End activity, some were happening concurrently, and some were ongoing. There is a time axis that very roughly shows activities that were initiated prior to the central activity (those above it), and those that were expected to follow it (those below it). Overlapping triangles very roughly indicate activities that overlapped in time. This approximation is very rough because the timing of these activities is not of great significance here. In particular, the chart does not show the complexity of the relationships between the activities that have just been described. And, it is absolutely not meant to convey the idea that the activities are performed in a waterfall-type fashion with

the output of one activity becoming the input for the next. Focus-On charts should ideally be drawn on larger spaces with the triangles repositioned and resized to more accurately reflect the more complex relationships that exist in a workplace so as not to erroneously suggest the workplace uses a waterfall process when it does not. All my depictions were severely constrained by space requirements and therefore unfortunately erroneously give the impression that the activities at an organization occurred in a waterfall-like fashion.

Reading the list of activities in a chart from top to bottom produces a narrative for the project that can provide a very rough picture of the work. For instance, Talia's chart begins with the ongoing Client Activities to Influence Config-it activity. These activities collectively attempted to influence the software to be more useful from the clients' perspective. Clients provided feedback on the currently released version of Config-it, and descriptions of transformed products that clients would find even more useful. This activity was followed by the Strategizing/Analyzing activity, which determined that Talia should redevelop Config-it. Next was the activity that planned this work, which was followed by the activity to manage the work as it was concurrently being accomplished by the Creating the Config-it Front End team. It was expected that once the front-end and back-end work was completed, the software would be deployed and maintained at the various client organizations and that the Config-it software would be put to use (i.e., end users would use it to configure services for client customers).

It is easy to imagine how activities that preceded the activity Creating the Config-it Front End might influence it. For instance, the Planning Config-it activity produced software requirements that aimed "to satisfy the client" and "to specify implementable software." The requirements were conveyed to the Config-it front-end team and, as a

result, these aims were in fact being attended to by the Config-it front-end team and were being realized in the Config-it front-end software.

Paradoxically, activities that followed the Creating the Config-it Front End activity were also influential because these activities were anticipated by the team or by the organization. So for example, the Deploying and Maintaining activity is an expected future activity, and therefore those who anticipate this activity (especially those who are likely to be engaged in it), try to influence the Creating the Config-it Front End activity so that the software is maintainable and deployable. Without such influences the Config-it front-end software might not necessarily be maintainable and deployable because the designer and developer's attention might not be focused on these objectives. At Talia, some of the developers on the front-end team expected to be deploying and maintaining the project in the future, so I had some direct knowledge of the important aims of this activity. However, aims of future activities were generally perceived aims, as explained earlier.

All of the activities depicted in the Focus-On chart were activities the Config-it front-end team discussed during my observation period. The chart shows how some of these activities actively influenced the attributes of the software that was being produced by the Creating the Config-it Front End team. The activities' aims are written in the triangles. Some of these triangles representing an activity have an arrow beginning from them and terminating on the central activity, to indicate there are mechanisms in place influencing the central activity to achieve the aims indicated in the triangle. For example, the Planning Config-it activity is concerned that the Creating the Config-it Front End activity produce software that is (a) completed with limited resources and that (b) satisfies the client. The mechanism for how the important aims of the activity are

addressed is indicated in the text below the activity's name. In this case, the Client activities to Influence Config-it are influential on the Config-it software because (a) the product owner who is on the Config-it team liaises with the clients, (b) the clients write feature requests, which the product owner reads, and (c) the clients write bug reports detailing problems in the previous version of the software, which the product owner also reads. Also, parts of the feature requests and some bug reports are used in the Creating the Config-it Front End activity itself.

In the cases where no artefact or advocate has been introduced into the Creating the Config-it Front End activity by another activity, or in anticipation of a future activity, this is indicated by the absence of an arced arrow between an outside activity and the central activity. The absence of an arced arrow suggests that there is a possible weakness in the system because there is a possible lack of connection between the two activities. In these cases nothing exists to ensure that the important objectives of the other activity, perceived or real, are addressed in the Creating the Config-it Front End activity.

For example, there is no arrow between the Using Config-it activity and the Creating the Config-it Front End activity. This indicates there are no artefacts or advocates connecting these two activities. Therefore, end users, who are perceived to have the largest stake in the product being usable, have no influence on the creating activity. The fact that no mechanisms exist to ensure this aim reduces the chances that the team will effectively attend to usability as an important product objective, even if their intention is to do their best in this regard.

Beyond identifying whether or not an artefact is present to ensure an aim, there is a considerable amount that could be said about the effectiveness of a particular artefact for achieving objectives, but this is not addressed by the charting mechanism. Similarly,

advocates, who are engaged in two separate activities to ensure certain aims are achieved in the software, can be effective or ineffective in the activity they are attempting to influence, depending on many factors. Both these discussion are beyond the scope of this analysis.

Having described this charting mechanism, I now fully describe the specific activities at Talia and the software objectives that were important to these activities. Knowledge of these preferences influenced the members of the Creating the Config-it Front-end team to achieve certain aims in the software that were important to other activities.

Focus on the Talia activity Creating the Config-it Front End

In the following analysis, I concentrate on activities that influenced the Creating the Config-it Front End activity in some way and describe how they did this. This description provides much detail on the advocates and artefacts than the Focus-On chart in Figure 4 provides.

1. Client Activities to Influence Config-it: The product must meet the business needs of the client. “Client feature requests” and especially “product backlogs” (including bug reports from the currently deployed Config-it software) were key artefacts for making requests to either modify the existing product or evolve the product. In this project the designer and the developers were made aware of customer feature requests largely through the product owner in face-to-face meetings, although they did have access to parts of the customer feature requests and the backlogs.
2. Strategizing/Analyzing: This activity’s motive is to understand what product is needed in the market. The analyst informed me that the reason for developing the

Config-it software was to increase the organization's competitiveness and to complete an important product line. There was no link between the strategizing activity and the Creating the Config-it Front End activity and no observable means in place to ensure the focus of the team on these aims.

3. Planning Config-it: The Config-it software is intended to have value in the market when completed. The analyst informed me that the product owner would have created a plan for the product that identified the resources required for the project. Further, the product owner, with the help of the senior developer, produced requirements documents to describe the product precisely. Scores of documents would describe this product. The first of these, the requirements document for the user interface, appeared only in the third sprint. The Scrum master informed me this should not have impacted the work of the designer and developer in the first two sprints because their work was "one step removed from client requirements." However, without this document during the first two sprints, the designer had had to determine what aspects of the previous version of the software should be carried forward. He had had to arrange a walk-through of the GUI software, and had had to do considerable research on the product on the side. The user interface requirement document, presented in the third sprint by the technical lead who was also a part of the team to create Config-it, confirmed the direction already being taken by the designer and developer. However, having this direction solidified in textual form appeared to help the team advance because it allowed for misunderstandings and differences to be aired and resolved, as I observed in the requirements review meeting.

4. Managing the Config-it Front End Project: The product was designed with limited resources, particularly people, and within a limited timeframe. The project management method being trialled at Talia was Scrum. Tasks were controlled and tracked through project tracking artefacts which were Microsoft Excel spreadsheets maintained by the Scrum master, rather than wall-mounted displays as in many Scrum projects. This was because the team was not co-located. Further, because it was hard to log in to the server and make changes, only the Scrum master kept the spreadsheet up to date. Collective responsibility for the project management artefacts was therefore reduced.
5. Creating the Config-it Back End: a separate software team existed to redevelop the code to serve as the back end for the new front end. However, I know little about this activity because the activity to create the front end had very limited contact with it. It's quite possible a very clear front end to back end interface existed and I just did not see it. Regardless, the focus of the Config-it activity was evidently not on the interface between the front and back ends, nor was it on the product in its entirety.
6. Deploying and Maintaining Config-it: A major goal of the project was to develop a product that minimized future development effort, which meant it was important the software be well-designed. The developers on the team were advocates for this future activity because they were possible candidates for the team that would deploy and maintain Config-it and because they had been on maintenance teams for other products. To ensure Config-it was deployable and maintainable, a development document for the framework software was written by the framework developer and was reviewed by the feature developers. This document contained a

high-level description of the software parts for the framework and how these parts would dynamically interact. However, the problem with the development documents was that only developers could read these plans because they referred to software architectures, software objects, messaging, and other elements only understood by developers. The designers, for example, could not read them.

7. Using Config-it activities: Dozens of large organizations around the world use this software product to configure services for their clients. The end users of the product have no contact with the Creating the Config-it Front End team, and therefore have no direct influence on the usability of the product. Usability is an aspect of the product that is important to end users, but there is no advocate from the user community addressing this product objective.

Common methods for influencing the Creating Config-it Front End activity included (a) establishing a separate activity to address the aim, (b) creating artefacts to ensure an aim is achieved, (c) ensuring that certain individuals bridged both activities by having them participate in both, (d) assigning an advocate on the team for a future activity, and e) adding someone to the team and making it their responsibility to ensure an objective is met.

The Focus-On Chart in Figure 4 depicts how the Creating the Config-it Front End activity is influenced by outside activities, and how the work of the designer and developer is being influenced to adopt certain attributes or objectives. Artefacts that enter the activity serve to focus the team on important objectives and prevent the team from creating software that might not address the particular objective. These influences help to address the limited attention spans of normal human beings, support thinking on

important objectives, help the team to remember progress to date on an issue, and help when balancing various objectives. In the sense that these other activities are applying a force on the Creating the Config-it Front End to influence this activity, they are seen to be in contradiction with it. This will affect the software that the designer and developer are creating.

This analysis reveals some of the complexity of ensuring that certain software aims are achieved in the Creating the Config-it Front End activity. A key strategy was having individuals and/or artefacts present in two activities. Another strategy was to introduce artefacts that ensured aims important to other activities were achieved. Some of the artefacts used in the Creating the Config-it Front End activity served to meet important aims of other activities, and served to influence the work of the designer and the developer so that the entire system of activities appeared to be working toward ensuring the final product was (a) the right product at the right time for the market given available resources; (b) made in the fewest person-years possible (i.e., low current costs, and timely); (c) low-maintenance (i.e., low future costs); and (d) useful (i.e., met client needs and requirements criteria).

However, the diagram also shows that some activities that could or should have had an influence, or should have been anticipated, actually had very little or no direct influence. An example of this is the Strategizing/Analyzing activity undertaken by the organization's strategists. This activity determined what product was needed in the market and why. In this case, analysts determined the need for a more competitive product, and the completion of an important product line; however, to my knowledge, neither of these motives was present in the Creating the Config-it Front End activity.

Further, no artefact existed that could be used to ensure the final product embodied these objectives.

The analysis also showed that the focus of the team, at least while I was there, was not directed toward the compatibility of the front end with the back end or attributes that had to do with the product in its entirety (like responsiveness). In addition, nothing in particular ensured usability from the perspective of the end users of the system who were engaged in the Using Config-it activity because there was no mechanism for influencing the Creating the Config-it activity from the Using Config-it activity, although usability was perceived to be an important aim of end users.

The details of this analysis were confirmed with the senior analyst who launched the project to the extent that this was possible.

Figure 4. Focus-On Chart for the Config-it Front End at Talia. Shown is the Creating the Config-it Front End activity in relation to other relevant activities in this multi-national organization.

Focus on the Jaba activity Creating and Maintaining PlayLearn-it games

I also observed at three other organizations and I now briefly describe, rather than analyze, the software-creation and software-management activities at these organizations, and the way other activities influenced the software-creation activity.

Jaba was a venture-capital funded organization. At Jaba I observed the work involved in creating a game that was intended to support specific learning outcomes. One year prior to my observation period I met with the CTO of this organization. By the time I had finished my observations a year later, the company had increased in size by about a factor of six, moved twice, and changed its name once. While at Jaba I interviewed the game designer, the content designer (who was also the learning designer, and project manager), and the developer.

I observed at Jaba over a 19-day period, entering the workplace six times, mainly when the work was geared toward the novel learning-game (PlayLearn-it), that I was tracking. On days when I did not visit, the team mainly worked on other less novel projects or on maintenance tasks for games they'd previously released. I focused on the activity of creating and maintaining PlayLearn-it, although this was really just a slice of a larger activity that involved work on developing and maintaining about 10 games (i.e., each game did not launch a separately organized activity). I observed the game designer, the content designer and the developer.

The game designer was responsible for the underlying game mechanic and was always the first to begin the work of creating a game. He took into consideration the end goal of the game, the learning material that should be covered, and the capabilities of the product team, to determine a suitable game mechanic such as rock-paper-scissors. The

game mechanic did not determine the look of the game at all, but it did determine how a player could win the game and it was like the underlying game logic. The game designer also determined the game motif, i.e., the idea around which the dynamic action of the game occurred, and which had more influence over the game's look and feel than the game mechanic. The motif proposed for the game I observed and which emerged part way through the design of the game was "a matching game." In this matching game, players had to repeatedly match characters to tasks in order to complete a complex overall objective. Players were rated for their choices, especially their overall efficiency in accomplishing their objective. They were given feedback to help them become more efficient.

The content designer was responsible for all of the dialogue and messages built into the game. This generally contained a lot of learning material and consequently the content designer was also the learning designer. The team also had a graphic designer and an artist.

The developer I observed integrated all of the input from the other designers and wrote any necessary code for the game. However, to complicate matters from my perspective, the developer, an avid gamer himself, also acted as a game designer, particularly when it came to designing interactive elements for the game.

Jaba was very unusual in that it had more designers than developers on the project, which was no doubt related to the application domain. All other projects had one or two designers and multiple developers, but this project had a game designer, a content designer, a graphic designer, an artist, and a usability specialist. Altogether on the team there were five designers, one developer and a part-time student developer. Also, as mentioned previously, one could easily count the developer as a sixth designer, especially

considering his contributions toward designing some of the significant interactive *widgets* of the game (interactive elements of the user interface), and the fact that he had had training in human-computer interaction in his computer science degree. The two developers used a gaming platform developed in the organization and were building this new game on top of that platform. Although the game was developed in a completely different style from any of the organization's previous games, they were not developing without a foundation for a game structure.

The days at Jaba were very different from the days at Talia. Usually each day had two or three impromptu meetings organized on the fly, which was possible in part because the team occupied a shared space and they had ready access to large, well-lit, and available meeting rooms. Between meetings, people worked at their desks, but there were frequent breaks or interruptions for work-related chats with other members of the project team. There was also quite a bit of fun injected into the days, with many in-jokes, humour at the beginnings and ends of every meeting, and social events like the sharing of a birthday cake or a memorable barbecue on Jaba's building's rooftop to celebrate an accomplishment in the organization.

I observed at 15 scheduled and informal meetings, six of which had both a designer and a developer present. Sixty percent of scheduled meetings, 30% of impromptu meetings, and 46% of chats involved both a designer and a developer, showing that Jaba made good use of the different meeting mechanisms for designer-developer interactions.

I identified four other activities related to the activity to create and maintain games. The motive of the Analyzing activity was to clarify the contract with the client so that the team would know when their work was complete. The motive of the Managing

Game Projects activity was to make a plan for the use of the organization's resources for the various games. The motive of the Various Client activities was to meet the business needs of the client organizations. And finally, the motive of the Gaming and Learning activities was to use the game to learn a necessary workplace skill. The client organization had in fact contracted Jaba to develop the game so that their employees would enjoy learning certain skills.

Other activity systems at Jaba influenced the activity of Creating and Maintaining Games at Jaba. These are depicted in Figure 5. A separate activity to analyze games was responsible for defining the games to be developed at a very high level. These descriptions amounted to a contract with the client, principally existing to define the end of the work. Another activity called Managing Gaming Projects, prioritized, made plans for, and tracked the work of half a dozen games simultaneously. Like at Talia, artefacts existed to influence, control, and track the work of the team that created and maintained the games.

Artefacts at Jaba were considerably more informal than artefacts in other organizations and more likely to be quick whiteboard sketches than professionally drawn images generated with software tools. Documents played a role, but there were essentially only two documents per game. One was the game design document that was internal to the Creating and Maintaining Games activity, but each designer was interested in reading only a fragment of this document. The other was a "content document" that specified the learning aspects of the game. The core team that worked on PlayLearn-it worked directly with a subject-matter expert (SME) to ensure the game would be valuable to the client, especially in terms of its content, and the content document was a primary means of ensuring this.

The ways that the activity Creating and Maintaining games was influenced by other activities is depicted in Figure 5, which shows how the team was directed toward certain objectives. Direction toward certain objectives was assured through the use of artefacts and also by having individuals participate in more than one activity, which is not an uncommon situation. This representation of the work shows the pressures on the Creating and Maintaining Games activity to move the PlayLearn-it game in a certain direction. So for instance, the Analyzing activity put pressure on the Creating and Maintaining Games activity to ensure the software met very high-level client specifications. These were spelled out in a product specifications document read by the team, but written for the client. The Managing Gaming Projects activity put pressure on the Creating and Maintaining Games activity to make sure PlayLearn-it was profitable and timely. A subject matter expert acted as an advocate for a collection of various client activities, and worked to ensure the game content included the correct material, and that the game would be useful to the client organization and other organizations that would eventually use it. A final activity, the Gaming and Learning activity, was significantly removed from the Creating and Maintaining Games activity. End-user activities were anticipated at the client organization, but also in many other organizations across Canada that the team Creating and Maintaining Games knew nothing about. There was no link between these activities and the Creating and Maintaining Games activity at Jaba.

Important aims at Jaba, as is shown by the Focus-On chart, were that (a) the product met the client's specifications, (b) the product was "profitable, (c) the release was timely, (d) that game content included the material that was meant to be taught through game play, and (e) that the game *was useful*. Aims that the team hoped to achieve, but that were not so well addressed, as evidenced by the lack of connection between the

Gaming and learning activities and the Creating PlayLearn-it activity were usability, and “the fit between the game and the gaming context.”

Figure 5. Focus-On chart for the PlayLearn-it game at Jaba. Shown is the Creating and Maintaining PlayLearn-it activity in relation to other relevant activities at this fully-funded venture company.

Focus on the Telespace activity Creating the Sell-it Front End

Telespace required a complete redesign of a core product they had been using for decades and so they had spent years planning this change, making decisions about the overall shape of the product, and engaging analysts to specify new features. Telespace was unique because they were not a software company and the product they were creating was for internal use. At the time I arrived, the Sell-it project had reached the point where enough was known about what was needed in the new product that it was time to create it. A whole new activity was therefore launched, office space was rented, and scores of contract employees augmented a core team of Telespace employees to create the application. This was the only organization whose product creation team was largely contract employees, and whose software development environment existed solely for the one project. The new team included 30 developers, 3 designers and 4 others.

At Telespace I interviewed the designers' manager, the designer and a person whose title was "designer-slash-developer," to use his own words, and also his official title at the company, but who I will refer to here as the *designoper* (rhymes with developer) to improve the readability of this document. The designopers implemented the look and feel of the user interface. There were four designopers at Telespace. I observed at Telespace over an 8-day period, entering the workplace on five of those days. Because I was focused on one particular feature, I was mainly interested in the activity of one designer, one designoper, and the developer assigned to that feature. I shadowed the designer and the designoper and observed their work on this feature.

I observed the creation of the reporting feature of Sell-it. This was one of many features identified as a part of the project. This feature allowed users to request and print about 80 different report types, setting as many as nine different options from dozens of

possible options, depending on the report type, to tailor their report to their needs. A major question was what exactly the designer should design for users to allow them to tailor their reports, and the answer to this depended on what the designoper and developer could implement. No one considered the option of designing every possible popup window, each one with a different combination of options, because this would have meant hundreds of repetitive designs. A more generalized design was obviously called for, but the issue of how much to generalize depended largely on the capabilities of the designoper and the tools he was using. This was the first time the designoper would not be reproducing the designs of the designer by creating an exact replica of the designer's drawing in software. A considerable amount of interpretation would be required by the designoper, and the designer would need to know more about the limitations of the designoper's tools and both needed to know more about the capabilities of the back end from the developer assigned to the feature.

The designers' manager was responsible for the overall design of the hardware and software for the project. He produced artefacts that provided the overall direction of the product and also liaised with the clients. He made high-level decisions about the priorities for the design team, the two designers and the four designopers who reported to him. The designers' manager worked part-time. When I observed him, he was drawing a layout for the product's keyboard. The designer I followed worked on multiple features, rather than on product-wide designs, and called herself an interaction designer. A sign posted prominently at the entrance of her cubicle let others know she had expertise as an information architect, interaction designer, usability engineer, visual designer, and prototype engineer. There were two other designers on the project, each responsible for designing the windows, pop-ups, and interaction flows for scores of features.

The designer I followed worked on up to 10 features at once, keeping track of each one using electronic folders, physical folders, and sticky notes. These reminded her of her progress and the feature's outstanding issues. The designers created high-fidelity *wireframes* of a set of screens for a feature, using PowerPoint to do this. Wireframes were drawings of screens that include information about connections between the screens. At one point designers also wrote documentation to describe the interaction between the user and the designed screens, but when I observed, this practice had been reduced to either annotating the PowerPoint slides or to foregoing descriptions altogether when features were considered "obvious."

The designoper I followed was one of four designopers. Developers consider designopers to be designers; designers considered designopers to be developers. The designoper received the wireframes, which were interactive PowerPoint slides, and which conveyed a little of the interface's behaviour. He then coded them, partly through a drag-and-drop interface, and partly by writing snippets of code to implement some of the behaviours required in the interface. A list of rules by his desk reminded him precisely of the designopers' responsibilities and those of the back-end developers, and clearly described the code in the team's repository that he could and could not touch. He organized the workload for the other designopers and was responsible for ensuring the product's interface could be realized in both English and French. He also "resolved communication problems" between the design team and the developers; he was the only designoper invited to the meetings of designers and the meetings of developers, and one of his tasks was to explain the views of one group to the other.

A single developer was assigned to the reporting feature, and he was responsible for writing the back end of the feature and connecting it to the front end. The designer and designoper I followed frequently interacted with this developer.

The days at Telespace were stressful, as people worked toward what was described to me as an arbitrary deadline that would be difficult to achieve (and in fact, they did not achieve it). Further, there was a significant amount of tension between the designers' team and the developers' team. The developers blamed the designers for slipped deadlines on almost every feature of Sell-it, which I witnessed at a weekly progress meeting of 25 developers. Similarly, there were tensions over the "power over the code" that the designopers had, which had been restricted by the developers. This had occurred because on at least one occasion the designopers had caused hold-ups when their code submissions resulted in delays and interfered with the developers' access to the software base that they considered to be theirs. From this treatment, I concluded that in the eyes of the developers, if the designopers were developers at all, they were second-class developers, even though they were roughly equivalent in terms of years of experience.

I observed 11 meetings, 6 were scheduled and held in boardrooms, and 5 were impromptu. Impromptu meetings were encounters that lasted half an hour or more and frequently moved between someone's desk and the lab space where the meeting participants could use a previous version of the Sell-it product to confirm current capabilities. Scheduled meetings appeared to deal with crises, such as transferring work when someone quit, rejecting a design, rejecting a feature in the current software, realigning project priorities, and addressing problematic process issues. Product creation work happened at people's desks, in working meetings, and through numerous chats. The

designer and designoper I observed were in the same group, but were physically separated by another designer who had the cubicle between them, and neither could hear the other when they were at their own desks. The designer, designoper, and developer communicated largely through informal chats. Telespace was the only site with a higher percentage of chats than scheduled or impromptu meetings. In total, I observed 32 such chats, 41% of these involved the designer and a designoper and frequently also the developer who worked on the reporting feature. Chatting was the only means the individuals involved in the reporting feature kept in touch.

I observed both the designer and the designoper working at their desks for prolonged periods on three occasions, and in every case the work was punctuated by numerous chats. Most of these chats began with some member of the ad hoc feature team (the designer, designoper or developer assigned to the feature) arriving at the desk of another member with a printout of a screen designed by the designer. Sometimes another type of printout would support the chat, such as a printout of the contents of the developer's database or a printout of the designer's spreadsheet, which contained a definitive list of reports and options.

Other activity systems related to the one for creating the Sell-it front end are depicted in Figure 6. The motive for the Specifying the Sell-it product activity was to specify the new features for Sell-it. The motive for the Managing the Sell-it project activity was planning the use of resources. The motive for the Creating the Sell-it Back End activity was implementing Sell-it's functionality. The motive for the Client Activities and Service Depot activities was installing, setting up, and maintaining the product and training staff. And finally, the motive for the Retail Activities, of which the Sell-it product would eventually be a part, was meeting the needs of the buying public.

Creating was not a single activity at Telespace, but was subdivided into Creating the Sell-it Front End and Creating the Sell-it Back end. This was the only organization that did not have a multidisciplinary project team because of their use of the waterfall methodology, although the ad hoc teams forming around features were multidisciplinary. This was also the reason the designopers did not fit in—because they did not clearly belong in one group or the other. In the waterfall methodology the product is specified by analysts, designed by the design team, and developed by the developers etc. As previously described, in lieu of multidisciplinary meetings, designers, designopers, developers and analysts spent a great deal of time checking information with each other at each other's desks. Printouts of screen designs mediated this work, and an enormous amount of paper and coloured ink was used for this purpose. A single manager who was present at only one meeting in the entire time I was there (which I did not attend) managed the entire activity. She preferred to do this “at arm's length,” through the developer's manager and the designer's manager and I never once saw her. The developers' manager managed the 25 developers, and the designers were frequently identified as the cause for missed deadlines at the developers' weekly status meeting. The designers' manager managed the two designers and the four designopers, but this design group had status meetings only when necessary, as happened once when the developers' manager wrote the design group a letter on official letterhead paper listing important changes he wanted. Figure 6 depicts significant influences that were present in designer-designoper-developer interactions and key advocates and artefacts that ensured this trio focused on the aims indicated by the curvy arrows. These were that the new feature existed, that it was timely, that it was correct, and complete, and that it was produced without disrupting the progress of the developers.

Figure 6. Focus-On chart for the Sell-it Front End at Telespace. Shown is the Creating the Sell-it Front End activity in relation to other activities in this service organization.

Focus on the Quadrow Activity Creating Calcul-it

Quadrow was a very stable organization. Product creation at Quadrow was a joint activity. Most of the projects were Web-based and not particularly large. However, the organization had very high standards for their software, so consequently Web and accessibility standards played a major role in the day-to-day activities of the team members. While at Quadrow, I interviewed and observed the designer, the developer and the designer's manager. The designer was involved in five projects simultaneously, including the Calcul-it one I was tracking, but he also acted as an advisor on two other projects during the time I observed him. He produced standards-compliant, high-fidelity interactive HTML prototypes populated with sample data. He was part of a small group of designers. The developer was involved in two projects, although I only saw him working on Calcul-it. He rewrote the HTML prototypes he received from the designer in JSP, a programming technology, retaining only the aspects of the designer's prototype that had to do with standards compliance, such as style sheets. The developer also wrote code to implement restrictions on data entry fields, the computational engine, and interfaces to databases.

I observed at Quadrow over a 6-day period, entering the workplace four times. While I observed the one project, I was particularly interested in, I also observed a considerable amount of work on other projects because of the nature of the designer's day. Over the two and a half days I observed the designer, he did a significant amount of work on two projects, advanced two others slightly, and helped fellow designers on another two. Over the day and a half I observed the developer, he worked on only the one project, although he was also responsible for a second project. I interpreted this as a rough measure of the complexity of the various tasks because both were diligent workers.

The project I observed was atypical in the amount of control the project's client, who was external to the organization, exerted over the design. She had contracted out the look of the product to another company, and she and her team also created mock-ups for the interface.

The days at Quadrow were very different depending on whether I was observing the designer or the developer. Both attended a stand-up meeting first thing in the morning where the project manager reviewed the status of tickets (small chunks of work) for between two to six of the 25 projects he was managing. To give an illustration of the typical work pattern at Quadrow for the designer, after the stand-up meeting on my first day of observation, the designer returned to his desk to work and made some progress on four different projects. He also engaged in four chats with colleagues to arrange a meeting and to pass on some information to do with a ticket. One prolonged chat with the designer's manager was a discussion of a standards-compliant way of implementing searches, during which he passed on information about a relevant Web site addressing this issue. A similar pattern of interaction was observed on other days. The developer's days were relatively less eventful with only one chat in one and a half days.

Much of the work within the Creating Calcul-it activity was coordinated through a tailored ticketing system that the designer and developer both loved. This system organized their work around written stories determined by the designer, which were descriptions of brief snippets of interaction small enough for the developer to implement over several days of work. The ticketing system collected together all the relevant artefacts for working on the ticket in one tidy bundle.

I observed 11 scheduled and impromptu meetings, 8 of which had a designer and a developer present, 2 of which I captured on video. Four to seven people attended stand-

up meetings with an average number of six. Two to eight people attended meetings to create Calcul-it with an average of six people. I observed the designer working at his desk for prolonged periods on four occasions and the designer on three occasions.

Creating Calcul-it was a single activity at Quadrow. The Managing project team, which consisted of other developers who were working on other projects and the project manager, met daily for a stand-up meeting that ensured the timeliness of all the projects and the right use of resources. The quality-assurance team existed to ensure quality software, addressing completeness, robustness, and standards compliance by running automated and nonautomated test suites on any Calcul-it software that was submitted to the corporate repository. A production team ensured that standards for maintainability of the software were met by applying organizational standards to the work of the designer and developer, and also by rejecting any software that breached those standards. Contact between the people involved in the Creating Calcul-it team and the Client activities team occurred through periodic meetings where the clients provided rough designs for the user interface. Their main aim was to track progress on the project. Clients ran their own user acceptance tests, and frequently presented the team with cases that they had not dealt with or had not considered. End-user activities influenced the Creating Calcul-it activity at Quadrow through focus groups and through end-user testing sessions. These were to be conducted when the Calcul-it interface was more completely designed. Figure 7 provides an indication of important advocates and artefacts that were used to direct the attention of the designer and developer toward certain aims. The important objectives of the software at Quadrow were that it was timely, usable, useful, attractive, maintainable, complete, robust, and standards compliant.

Figure 7. Focus-On chart for Calcul-it at Quadrow. Shown is the Creating Calcul-it activity in relation to other activities in this well-established organization.

Summary

My goal was not to judge the observed software creation activities because I did not see them to completion and because each organization's circumstances differed. However, there was substantial evidence regarding each project's success, which I learned about when I informally followed up with my participants after I had received news their software was released. All teams completed their projects, their clients received their software, and three of the four software applications described in this chapter were released and are in use. The fourth one unfortunately was obsolete before it could be used because an assumption that had been made about technology was no longer relevant. Another indicator of the relevance and success of the projects was the number of end users of the three products in use, which ranged from hundreds to tens of thousands of people.

My goal in this section was to reflect on the mediators in designer-developer interactions in real software development environments that enter the activity from outside the activity. I began by identifying the software-creation activity at each workplace and then all activities related to this activity. I was interested in how these activities ensured that the designer and developer focused on certain objectives. Focus-On charts, which I created as an output of the analysis, and an extension to activity systems analysis, showed exactly how the project work was influenced from *outside* the activity. They showed that software development work and the work surrounding it is structured to ensure that those who develop software—the designer and developer—focus on certain software objectives. Although Focus-On charts are imperfect representations of the work, if considered from an objective perspective (generating a more objective representation would require observing all of the activities), the charts brought important

elements to the fore and provided a quick picture of project priorities and the social and cultural influences on the designer and developer as seen through an examination of their work, and as experienced by them.

Certain strategies were employed at all of the organizations to ensure that attention was paid to valued objectives. Very important objectives had entire activity systems dedicated to their achievement. The best examples were the management activities that ensured the software was right for the market, was produced with minimal resources, and was on time. A common strategy of such activities was introducing an artefact into the creating activity. These helped to focus the designer's attention on important software attributes. Another common strategy was to have the same individuals participate in multiple activities (such as creating the software and managing the creation of the software).

The Focus-On charts not only revealed important objectives and the means used to achieve them, they also revealed some areas that projects were not focusing on. This is not to say that this was necessarily a problem for the organization because there could be many reasons for not focusing on a potential objective. However, activities that were related to the Creating the Config-it Front End activity that did not have shared artefacts or advocates with this activity did raise questions. The issue here was whether or not the software that was to be created would be suitable for, or valued by, this other activity, because certain aims of potential value to those other activities may not have been achieved.

Concluding Statements

This chapter began with the question, “How is the work of designers and developers structured within organizations and what role do artefacts play here?” The teams involved in product creation work organized themselves differently, and used a broad range of different artefacts. However, much can be said about the structure of this work and the use of artefacts that applies across all projects.

The concept of an activity system was used consistently throughout this analysis to structure the data from the observations. Using this concept I was able to distinguish one activity from another by looking for, and then describing, the structure I found in the work. The structure I looked for was mainly derived from consistent patterns in the ways that humans coordinate challenging work that is accomplished collectively. The consistent patterns reflect natural, but evolved, human abilities to organize work (Leontiev, 1981).

I found that work was organized quite differently at various sites. In general, though, designers and developers tended to interact through two different activities. They were part of an organized multidisciplinary activity to create software, and also part of a separately organized, multidisciplinary activity to manage the creation of software. An activity for creating software was usually organized around the creation of a single software project, but at one site, Jaba, the software team as a whole worked on multiple projects simultaneously and continuously, both creating and maintaining a score of learning games.

While traditionally developers create the code that produces the software, this study found that modern tools increasingly allow technology-savvy designers to create the software as well. At Talia, the code for the software’s interface, at least the visual

part, was eventually generated directly by the designers' actions, and not the developers, due to a change in the developers' toolset, which included a tool the designer could use. At Quadrow, essential elements that ensured accessibility and the look were carried over directly from the designer's work. At Telespace, because designers did not have the skills or time to master the necessary tools, the designoper (a new role historically speaking) generated code for the look and certain aspects of the feel of their user interface, and also acted as a bridge between the designer and back-end developer. At Jaba, designer-generated art, graphics and gaming content (mostly game character dialogue) were a significant part of the game and were incorporated into the gaming software directly. Though, strictly speaking, not code, these artefacts became a part of the software when the developer integrated them with the code.

At all sites the boundary between the work of the designer and developer was negotiated, and evolved through informal and indirect changes in practices. Part of the reason for this evolution was the changing skill set of the designer. For instance, at Talia the designer garnered expertise in a new tool and changed the developer's expected tasks. At Quadrow, the designer (who had worked as a developer on other projects) was continually striving to produce realistic and interactive prototypes that were close to being the front end of the product under development and, as he did so, the developer's work evolved accordingly. However, the ambiguity in the work's boundary was not always because of the designer's changing tool set. At Jaba, the person who was the recognized developer had become an accepted designer of gaming widgets. At Telespace, the designoper fulfilled an intermediary and temporary role bridging the designer and developer's role as the designers did not have the skills to use a particular tool. The

boundary between the roles therefore appears to be negotiated on a project-by-project basis and can shift throughout the course of a project.

Managing software projects varied even more widely across organizations. However, in all cases the very existence of this activity indicated the widespread importance that software be completed on time and with a minimal amount of resources.

Looking more closely at the creating activity at Talia, I focused on elements that provided direction for the work. I identified these as the motives (the business reasons for the activity), outcomes (desired results) and objectives (desired qualities of the software). These aims I saw as forces in potential contradiction with one another. They had differing priorities and were balanced by the product creation team. Important motives, outcomes or objectives were ensured by a number of strategies. For very important motives, a person was assigned to a role and made responsible for the aim.

Within software-creation activities, artefacts helped to ensure motives were realized, to understand the object of the activity, to reveal objectives, and to resolve contradictions, particularly the forces on the creating activity that ensured the software that was produced achieved certain valued aims.

The software that was the object of the creating activities was a special artefact because of its unique qualities that have been previously discussed. A variety of artefacts were mentioned. Many of these were tangible, such as mock-ups. Some artefacts, such as exemplars, were mentioned in the talk between designers and developers (e.g., “like XYZ”). Some were conceptual artefacts (e.g., “it’s a content management system”). Meetings were considered to be an especially useful artefact or tool of the team. Meetings were a place where important adjustments were made in people’s thinking about the object and aims of the activity. They were a place where contradictions between motives,

objectives or outcomes were addressed; meetings will be discussed in detail in Perspective 4.

Creating a new activity was one way important aims were ensured so I therefore also used activity systems analysis to study surrounding activities. At each site I developed a Focus-On Chart of activities that were related to the create activity, some of which had occurred, were underway, were ongoing, or would happen in the future. This analysis of surrounding activities explained the reason for some important artefacts that entered into designer-developer interactions and the presence of some of the members on the creating teams. It also explained how the attention of designers was directed toward certain aims and not others.

Two strategies were commonly used to connect other activities and the create activity in order to achieve aims. One way was for an individual to engage in the create activity and another activity. The other way was to create and use artefacts to bridge the two activities. However, the analysis also pointed out that some aims (perceived or real) that the designer and developer were addressing, were associated with fewer mechanisms to ensure their achievement. These aims were generally not represented in the creating activity by an artefact, and they had no advocate.

The working out of various aims, both internal and external to the software-creation activity through advocates and artefacts was the continuous theme that linked the detailed study of Talia's creating and managing activities, and the studies of the interrelated activity systems at the four sites. This working through of aims was an important accomplishment of designer-developer interactions.

Results and Discussion Perspective 3: The Meaning of Mediated Interactions

Acquiring a more subjective perspective on an activity can provide additional insights that are not possible to achieve from merely observing an activity. The goal of this chapter is to answer the question: What meaning do artefact-mediated interactions have for designers and developers? To this end, 26 interviews about work guided by two interview scripts, were conducted with 21 designers and developers. Twenty of these interviews were analyzed using grounded theory.

Approach to the Analysis

Grounded theory was adapted for use within cultural-historical psychology to analyze interview data. When coding, the development of substantive codes and categories was influenced by theoretical concepts in cultural-historical psychology (e.g., goals, contradictions, etc.), rather than the concepts of Strauss's (1993) theory of action as in Strauss and Corbin's version of grounded theory. Appendix E summarizes the 10 steps that were followed in the analysis and described in detail below.

Grounded Theory Adapted for Cultural-Historical Psychology

Step 1, open coding. To begin, I familiarized myself with the transcripts. Then I analyzed each transcript in turn. Using open coding I went through each interview initially coding by phrases, then sentences, then even larger chunks that included multiple turns in the dialogue, each time asking: What is this about? This process resulted in the production of approximately 50 activity-theory influenced concepts under the following broad categories: Artefacts, Joint Activity, Expressions of Emotions, Development of a User-Interface Framework, Goals, and Tensions. Throughout this step, many codes, and their corresponding coded quotes, were combined and split as the need arose.

Step 2, selective coding. Selective coding eliminated many codes, merged more codes, and identified an important set of codes that reflected the participants' views of software-creation work. Fifteen concepts emerged: participant's Personal Goals for the Project, View of Who Determines the User Interface, View of Design Artefacts Generally, View of Artefacts in the Project, View of Artefacts in Designer-Developer Interactions, View of Contradictions/Tensions Generally, View of Contradictions/Tensions in the Project, View of Contradictions/Tensions in Designer-Developer Interactions, Project Goals, Expressions of Emotions, View of Designers, View of Developers, View of the Business Person, Problematic Artefacts, and View of the Design Process.

As an example of this process, responses to a question about tensions resulted in participants making comments about contradictions/tensions in general, about key contradictions in the project, and about contradictions in designer-developer interactions. Designer Owen, for example, discussed the nature of tensions in the workplace relating to the creation of the PlayLearn-it game 10 times. This was a typical quote:

And so that, at that point, that's when I know. So what had started off as a [Type 1 gaming] method and then through this painful process—and it's brutal, I mean, it...it hurts; I have sleepless nights, the team has sleepless nights because there's a lot of tension—through this process, something new and different has emerged. And that new and different thing is this...new game, this [Type 2 gaming] method.

This quote and nine others discussed tensions and informed my understanding of Owen's view of tension and its place in the project work.

The concepts View of Design Artefacts Generally and Who Owned the User Interface emerged indirectly through participants' comments throughout the interview. In contrast, other concepts, such as Personal Goals for the Project, emerged as a result of a

direct question, although the topic also came up naturally in other parts of the interview. All the concepts came from what participants said, which was shaped by open-ended interview questions whose content was influenced by cultural-historical psychology.

Between 2 and 30 quotes were collected for each concept from each participant, averaging 10 quotes per concept per participant. This meant that more than 1600 quotes (10 quotes x 8 concepts x 20 participants) informed the work of building the model, with some quotes relevant to multiple concepts.

Step 3, writing memos. Memos, one for each participant (rather than one per concept as is typical in grounded theory) were organized by the 15 concepts previously mentioned. This allowed me to explore each participant's perspective across a broad range of concepts. To give an example of the memo content, one of the items in each memo was Contradictions/Tensions. For each quote, I noted the forces at work in the tension and, if relevant, aids for relieving the tension. For example, in Owen's memo I wrote

Contradictions/Tensions mentioned by Owen:

No game versus a new game (resolution: the design process, which is very "painful," but produces a new game)

Step 4, summarizing memos. Findings in each memo were then summarized for each concept. For example, the 10 quotes from Owen revealed he regarded a tension as a positive learning opportunity for the team in the sense that they were an impetus to designing the game or a necessity for transforming the work, and this became my understanding of his View of Contradictions/Tensions Generally.

Processing further quotations about tensions led to the conclusion that Owen was primarily concerned about tensions within the team. He mentioned no tensions between

himself and the developer or any other team member. I therefore wrote that his concerns related to “integrating the contributions of all team members and keeping the design process going versus ‘stagnating’ the flow of ideas, by keeping the team motivated.” This became my summary of his View of Contradictions/Tensions in the Project. My summary of View of Contradictions/Tensions in Designer-Developer Interactions became “same as project contradictions,” because Owen did not experience tensions in his relationship with the developer, but he did view the developer as a member of the team and was therefore concerned about his motivation level. However, even though I summarized my findings for each concept in my memos, each memo was getting lengthy.

Step 5, visualizing summarized findings. A spreadsheet was created to contain all of the summarized findings using rows for participants and columns for concepts. For example, the summarizing phrase “tensions are a learning opportunity, an impetus to designing the game” became an entry in a cell under the column View of Contradictions/Tensions Generally in Owen’s row. The completed spreadsheet, which was 10 pages long, enabled a view of the data as a whole.

Step 6, looking for relationships in the data. The completed spreadsheet revealed similarities between the responses for a subset of concepts for a given participant. For each row, the responses to 8 of the 15 concepts were strongly related to each other because they shared a common quality.

Other concepts were not saturated, such as Problematic Artefacts and Emotions. View of the Design Process concept had many quotes, but this concept captured mainly factual quotes on how the software product was created, so it did not reveal participants’ views. Other categories, such as View of Designers, View of Developers, and View of the Business Person, contained quotations that expressed stereotypical views of other

professions, but did not reveal insights into how participants viewed their work with these other specialists.

Table 15 *Key Concepts Contributing to the Category Interactional Identity.*

Higher-level category	Category	Concept
Interactional Identity	Goals	Personal Goals for the Project
	Interface	View of Who Determines the User Interface
	Artefacts	View of Design Artefacts Generally View of Artefacts in the Project View of Artefacts in Designer-Developer Interactions
	Contradictions/ Tensions	View of Contradictions/Tensions Generally View of Contradictions/Tensions in the Project View of Contradictions/Tensions in Designer-Developer Interactions

Step 7, abstracting. I then focused on the eight concepts listed in Table 15. I grouped similar concepts together and created simpler names. The category Goals was created with one concept in it, namely Personal Goals for the Project. Similarly, the category Interface was created with one concept in it: View of Who Determines the User Interface. A single higher-level category called Artefacts was created to group together the three concepts having to do with views of artefacts. Similarly, a single higher-level category called Contradictions/Tensions was created to group together the three concepts involving views of contradictions. Then a single higher-level category was created from the four resulting categories: Goals, Interface, Artefacts, and Contradictions/Tensions. Therefore, one theoretical category emerged from eight substantive concepts through the method of constant comparison (i.e., by comparing items and grouping similar items

together). What bound all of the concepts listed in Table 15 together to allow them to be grouped into a single higher-level category was that they all captured the participants' views about interacting within the team.

For each individual, the similarity between the responses in the four categories was an expression of the participant's identity in the context of their jointly experienced mediated work on the software-creation team. The higher-level category Interactional Identity captured how a team member saw himself or herself in interaction with others on the team. (Another name considered for this category was Identity in Interaction because the category is not meant to be about an identity located in an individual, but in an individual's interactions with others.) The responses in each of the four categories individually and collectively expressed how the participant approached their design or development work with the other team members. The four categories revealed an identity, inner story, or script that directed the individual's approach to executing their role as a designer or a developer when interacting with other team members. It expressed how a person saw themselves in interactions and, of particular importance to this research, it revealed their overall approach to using artefacts as mediators.

Each of the eight concepts revealed an individual's Interactional Identity because each concept captured some aspect of an individual's view of team interactions. The category Goals revealed the participant's specific objectives relative to others on the team. The category Interface revealed views about the team's shared object and differential, overlapping, or competing ownership of it. The category Artefact captured views on artefacts generally, on the project team, and in designer-developer interactions. The category Contradictions/Tensions captured views on contradictions and tensions generally, on the project team, and in designer-developer interactions.

Step 8, reflecting. Hypothesizing what this category might mean relative to other categories in cultural-historical psychology, it was clear that in all cases an Interactional Identity did not negate an individual's self-identification as a designer or a developer. Rather, an Interactional Identity appeared as something that augmented an individual's work identity.

For all participants, work identity (i.e., "I am a designer" or "I am a developer") was strongly linked to roles (i.e., "designing the user interface" or "developing the software"). However, roles are limiting because they specify an individual's responsibilities vis-à-vis the software product, but do not prescribe how to execute those responsibilities. A role also does not prescribe how to relate to others on a project. The results of this study suggest that this latter gap is what the abstract category Interactional Identity addresses, that is, this category uncovers an identity that determines how an individual approaches their interactions with others on a team.

Interactional Identity, however, is not only an extension of a work identity, but also an aspect of personality understood in an activity theoretical sense (Leontiev, 1978b) as opposed to a collection of stable traits. While work identities are associated with socially prescribed roles, the Interactional Identity appears to be more closely determined by the individual themselves as a response to who they are, the activity they are engaged in, and their past experiences.

Step 9, verification. To verify these ideas, each participant's summarized data was examined for each of the eight concepts to find a phrase that captured that individual's unique orientation to the way he or she interacted within the team, and especially with the designer or developer with whom they worked. Search functions, dictionaries and thesauri were an aid to finding phrases. The phrase that was selected was the one that best

captured the participant's orientation toward team interactions. For example, the developer whose View of Design Artefacts Generally was that they were "a means to an end" or "cognitive aids to problem solving" turned out to have a very pragmatic orientation in other areas too. He viewed Contradictions/Tensions as a good thing because they "pushed the envelope," and his Personal Goals for the Project was "to leverage the creativity of others to produce an application that performs well and provides the necessary functionality as determined by others." Each concept therefore consistently revealed a developer who was pragmatic in his interactions. This person was very focused on end results and his interactions with others, including the designer, were about producing the software product. He related to others in terms of their role in the production of the product and he depended on them to fulfill their roles. The phrase "one who aims to discern and meet achievable software goals" effectively described his Interactional Identity. This identity was unique and no other developer in this study shared it. I then truncated the phrase into a single word or two, which I used as a handle. In this case I chose the handle pragmatist.

Step 10, axial coding. Four categories were directly related to the higher-level category Interactional Identity. These were (a) Goals (b) Interface, (c) View of Design Artefacts Generally, and (d) View of Contradictions/Tensions Generally. The other four concepts were related indirectly, two through (c) and two through (d). Connecting my spreadsheet data with demographic data, certain factors that influenced Interactional Identity emerged such as education. A model of Interactional Identity was generated from these observations.

An exploration of individual differences emerged from the analysis. I concluded that the way to finding the meaning behind the use of artefacts in designer-developer

interactions from the perspective of an individual designer or developer was to capture the individual's Interactional Identity because this revealed their unique orientation toward artefact use when interacting.

The next section describes two Interactional Identities that were found and justifies them using evidence from transcripts. The individual Interactional Identities also revealed much about the designers and developers. The analysis and justification for deriving an Interactional Identity is accomplished using quotes associated with the concepts and categories from Table 15.

How Designers and Developers See Their Work

Full Case Study: Designer Owen's Interactional Identity

This section contains a detailed case study of designer Owen because he had a clear and colourful identity. In order to show how the category Interactional Identity was derived from the categories shown in Table 15, I review each category and then supply some of the quotes from each category that were made by Owen and collected together in that category.

Owen was a lead game designer and chief technology officer in his organization. He had 10 ½ years of experience in design work. He was one of five people creating a game for an established client. To identify Owen's Interactional Identity, I studied the quotes in each of the categories listed in Table 15. Below I provide the evidence from each of the categories that led to the conclusion that Owen's Interactional Identity was *movie director*. In order to arrive at an Interactional Identity, I required that all categories have evidence, in the form of quotes, that supported the particular Interactional Identity. So, the identity had to "shine through" in every category.

Goals. One of Owen's goals in his work as lead game designer was to find “the central motif” for the game, a word also used in theatre to describe something that motivates and brings the team together. As Owen said, “The main thing in design is the concept, or the motif, and at this point I have no idea what that's going to be” (Owen).

Another of Owen's goals was to motivate people so that they cared about their work. The way he did this was the way he felt a good movie director might work. He encouraged the team to explore. Also, on occasion he would deliberately make himself look foolish to allow others to look better, a strategy he considered a good movie director would employ (notice also how he accidentally calls a team member an “actor” in the quote below).

You often have to, um, give...help them [the team] make...you know, ... break out of their shell by doing it yourself first. So rather...You know, like that example I gave, “Do it really, really loud, and then do it really, really soft.” [This is what a director might say to actors to get them to explore a range of expression.] If I do that, and I take the risk of looking foolish, then you're [someone on the team is] more comfortable. If I don't think I'm getting enough, if I [as a movie director] come out and I take it really, really loud and really, really big and make a real fool out of myself in front of everyone, it's easier for the actor [someone on the design team] to do that. I found that worked really, really well, working even on the design [of the game].

Interface. Owen struggles with his ego to allow others to contribute to the design of the games because his instinct is that he should determine every part of it. As a game designer (he designs the ludic elements), Owen would prefer to work in isolation, but he needs the team to give him feedback and to provide other parts of the game (e.g., the content, the writing, the code, the art, and the graphics), much like a director cannot turn a script into a play without the contributions of a cast of actors.

My personal shortcoming is I suffer a lot from NIH: Not Invented Here. OK. ... It's something I struggle with constantly. ... What usually happens is when someone says an idea, my first reaction, internal reaction, is That's not good and here's why. And I have to stomp on that reaction. I—if you notice, whenever

someone comes with a new idea, I pause. I then say back to them the idea that they just said. I'm doing this because I'm having to internally count to 10 and not stomp on it.

Owen believes that, as a game designer, he must hold an opinion about everything (as he says a director would in the case of a movie) or people seeking feedback would get frustrated. So, he relinquishes control over parts of the game and “offers opinions.” If you don’t offer an opinion, “they get frustrated”. “It’s [offering an opinion about everything is] very similar to what a movie director has to do.” “I learned this, actually, believe it or not, when I was in film school.” Owen explained this as follows:

The director has to have an opinion. If the director comes in and says, “Oh, just give me something groovy.”—an expression I learned from my film school days—that’s what you don’t wanna do [is give vague directions]. “Yeah! Give me something groovy.” [give a vague direction.] “Yeah! I want—just move the cam[era]” [give a specific direction.] —they [directors] have to have an opinion about everything. And that’s why they’re called a director. [They] direct every little thing. But they can’t do anything. Ah, the same thing seems to happen in game design.

View of Artefact Generally. Owen sometimes sees artefacts as props in a drama.

He told me of an occasion when he literally organized a game re-enactment in the office with the designer, writer, and developer to explore variations for the “voice” of the gaming characters.

So we put together a simple prototype concept. Just take the [gaming] characters around the [real] office [this worked because the setting of the game was a virtual office], and [we] do some silly event around them [the characters]. ... So we had me and [the developer] and [the writer] and [the graphic designer]. They [all the characters in the game] were all in there [in the reenactment]. [The graphic designer] just drew them [the characters] and—it was fun. Um, and then I told [the writer], I said, just try different voices, ways of doing each of the things for each of the people. You know, don’t worry about being consistent; just expand. So she—that let her try 6 or 7 different things, constructing each, and then we could play them [literally acting the game out in the real office setting]. Then after playing them she said, “This one’s really working well for me.” And we converged on that, we picked that.

For Owen the drama and the flow of interactions between people were more important to creating a great game than the artefacts used in the workplace. For instance, Owen told a story of how creating a new game from another great game just does not work, as he learned when designing his first game. For the new game design, the “magic” is not there.

I copied it pixel for pixel. I did—I looked at the source code, I copied some things line by line, and it still doesn't feel the same. Why? You know, it's a great discovery to have. That's an important discovery. We copied it—you know, so I copied it and then [our writer] wrote the text. And we put it together. We're never gonna use it.

View of Artefacts in the Project. Owen sees artefacts as being props in a production and elements around which drama unfolds. For Owen a key artefact in the project is the game design document. Owen regularly writes a game design document to describe the ludic elements of the game and aspects of the game framework. The game design document is the central design document from which the team works and could be compared to a script because this is the artefact around which the activity is initially organized and from which the game emerges in a painful process involving the entire team. Here Owen describes how the game design document comes into existence and how he uses it.

by starting with something concrete [another game]—I start adding and modifying and taking away and playing with it until, um, I think I've got something that is starting to cover off what has to happen [in the game I have to design]. At that point, I write it down in a document, a design document. ... And then I start to share it with the team ... And there's a lot of pain and gnashing of teeth....

Owen treats the design document like a movie director might treat a script. It is something around which others rally. “Everyone kinda agrees to the design document.”

Another artefact, “working meetings,” were like rehearsals of the gamer playing the game because the team typically ran through a typical game play at their meetings. This was a recurring pattern I saw many times at Jaba, although the team did not appear to be consciously aware they were working this way in that none of them mentioned this when I interviewed them about their process. Working meetings were experienced as chaotic explorations of the current game and its shortcomings, rather than an ordered exploration of the gaming product under development.

Owen preferred informal, open meetings that were like chats.

I prefer, when I'm working, ... sitting down with a designer, sitting down with someone and working with them in an open environment, because it drags people in. They say—what I see happens is they go, ‘That's something that I have an opinion about.’ And if someone has an opinion about something that means they care about it. And if they care about it, you know, then they give good ideas and they're gonna contribute to it.

For Owen, the artefact, which was the game itself, was like a play being produced. The game has all the elements of a play including characters, action, sequences of events and an ending. He talks about the details that come out through interaction with the team members. Here he talks about the contribution of the content designer: “Exactly what are we gonna cover and when? What's the content that's gonna go there? What are the events? What are the characters? What are the ‘this’es, what are the ‘that’s’?”

Contradictions/Tensions. Owen sees contradictions or tensions as learning opportunities. For example, Owen described a tension between the contents of his game design document, which contained the game's mechanics, and the team's understanding of it. His view was that resolving this tension resulted in the creation of a ‘motif’ (i.e., the central idea of the game) that, once identified, would motivate the team to continue their work.

We now get this tension, because for some reason they feel that what I wrote down in my first draft [of the design document] is...cold or brilliant. ... And they don't get it. And because they don't get it, they struggle and try to... understand it. And they're not. 'Cause there's a lot of problems with it. ... through the iteration and through the discussion, my ideas start to clarify. And at some point, what magically emerges is this concept, or this motif, that everyone suddenly grabs onto and gets. And it becomes this unifying aspect of the method that everyone rallies behind.

Also, Owen's theory about design in general is that it is inherently imbued with drama and tension (i.e., contradictions). For instance, he described design work in terms of cycles of expansion and contraction. And, he sees the work of creating games as a dramatic process that has a built-in tension between the forces of expansion and contraction.

Initially everything expands, expands, expands, expands, expands, as new ideas get thrown in the mix. And I keep trying to incorporate those ideas. I know...I know I'm on the right track when I start taking things away. As soon as my mental framework moves from adding to taking things away, I know it's near finished.

At other times Owen discussed design work in terms of cycles of abstraction and concretization (two other forces, one other tension). "There's holes, it's horrible. And at this point what I try to do, is we try to start reducing it down to something concrete again. So I started concrete, went abstract, and then go down to concrete."

No other designer spoke using comparable dramatic terms.

View of Contradictions/Tensions in the Project. There is a strong connection between Owen's View of Contradictions/Tensions Generally, which is that they are learning opportunities and an inherent element of design work, and his View of Contradictions/Tensions in the Project. As the lead designer, Owen regarded the major contradiction in the project to be the need to keep the staff/cast motivated as opposed to unhappy and unproductive. The meetings, the game itself (in whatever state), and the design document, all helped resolve this contradiction. The other major contradiction

Owen experienced was integrating the work of all of the designers/cast (including the developer who was also a game designer) given Owen's view that each designer pulled the game in a different direction.

Coming to the conclusion that Owen's Interactional Identity was movie director was straightforward once I had distilled all of his information into the spreadsheet; however, for most participants it was much harder to come up with a phrase that captured how they uniquely perceived their interactions with others. Owen all but told me his Interactional Identity when he discussed his approach to his work:

I've seen it [the design process] in film, as a director, and designing—you know, 'cause that's just a design problem. I've seen it in hardware, designing laser chips. And now I'm seeing it in games, software. I've seen it in three different things, and it's absolutely fascinating 'cause everything I've learned abstractly applies. It's the same. I take things I've learned from...film, and hardware, and they're applicable here as well.

However, even though Owen suggested his Interactional Identity to me, I still had to confirm his Interactional Identity as movie director because I had to make sure it came across in all the quotes associated with the coded categories found in Table 15. The alternative identity of “laser chip designer,” which he also mentioned in the quote above, did not carry through any of the categories. The Interactional Identities of all but a couple of other participants were derived by “reading between the lines” or extrapolating from the evidence. Owen's case was chosen as an example to explain how Owen's data, captured by the eight interrelated concepts, revealed and reflected Owen's Interactional Identity. This example was clear and was the easiest to derive, although not at all obvious at the beginning of the analysis, and certainly not an identity anyone would have attributed to Owen with little or even significant amounts of thought. The identification of the identity required careful examination of the right data from the interview. The fact

that Owen himself all but suggested his Interactional Identity in the course of explaining his work as a designer adds weight to the evidence that such an abstract category exists.

The reasons I came to understand Owen as viewing himself as a movie director are summarized in Figure 8.

Figure 8. A model of designer Owen's Interactional Identity movie director.

The model shows the seven concepts and the higher-level categories that emerged from the analysis with specific information from Owen's case written below the concept and category name. The model shows how the Interactional Identity movie director was derived from four other categories, namely, Goals, Interface, Artefacts, and Contradictions/Tensions, hence the arrow from the four categories to the Interactional Identity box. The arrow in the other direction, from the designer's Interactional Identity to the four elements, shows that the Interactional Identity encapsulates the other four aspects. The Interactional Identity movie director captures the unique way that Owen brings meaning to his work, his mediated interactions with project team members, including the developer, and his view of workplace contradictions or tensions that must be collectively resolved.

Following downwards, the model shows a strong connection between a person's View of Design Artefacts Generally, their View of Artefacts in the Project, and their View of Artefacts in Designer-Developer Interactions. There is also a strong connection between the categories of View of Contradictions/Tensions Generally, View of Contradictions/Tensions in the Project, and View of Contradictions/Tensions in Designer-Developer Interactions.

The diagram depicts the two influences on Interactional Identity observed in this study that were consistent across designers. The first is education. The fact that Owen has post-secondary education in film and that he uses the role of director to describe how he does design work is not a coincidence. The second influence is Circumstances. Owen would not have been able to assume the identity of a director had he not been in a leadership role on the project. Also, his role is tied to the particular type of product he is designing—he says that designing like a movie director happens because he is designing

a game. Finally, he maintains the Interactional Identity because it is effective in producing games.

In summary, while Owen's case is typical in terms of the relationships that were observed across the eight coded categories, it is also unique in that no other designer understood their work in the same way. This analysis has found that to understand how a person uniquely uses artefacts in designer-developer interactions one requires an understanding of a person's Interactional Identity, which effectively captures how an individual uses artefacts generally.

Owen as movie director is my best explanation of how Owen presented himself as a designer during his interview. However, I also observed Owen for hours at his work and in a dozen meetings, and these observations corroborated this understanding of his behaviour as well.

This analysis of Owen as movie director is an in-depth, holistic, dynamic, and individualistic understanding of a particular individual's talk- and artefact-mediated interactions with others on the team, including the developer. This analysis says what collections of artefacts mean to an individual engaged in activity. In this case, Owen uses artefacts in his work with others as a movie director would use artefacts in his work with others, and he interacts with the team in the way a movie director would interact with his cast and crew without, incidentally, exposing his identity overtly. An Interactional Identity says exactly how an individual gives meaning to their artefact-mediated interactions with others on the team.

Full Case Study: Developer Wil's Interactional Identity

This second full case study is of a developer, Wil, who was one of three developers on the project team at Quadrow and who was responsible for implementing

the application, supervising another programmer, and liaising with a third programmer who was designing a database for the back end. He had had 22 years of experience as a programmer. He sees his work in terms of binding.

I had observed Wil for a week, but it was the interview that revealed Wil's Interactional Identity. When I interviewed Wil he had begun developing the application his team was working on, but there was much work left to be completed and a huge amount of change was happening with respect to the design of the user interface. There were many tensions in the team's interactions with the client. In the interview Wil presented himself as a *binder*, someone whose actions cause two elements to cohere or stick together in a mass. In Wil's case the two elements to be bound were a variety of different artefacts (e.g., a mock-up of the user interface or a story of user interaction) and the glue that caused the two elements to stick together was the program code he wrote and rigorously tested using a suite of test cases he also coded. The following outlines key pieces of evidence from each of the eight concepts that individually and collectively support the claim that Wil's Interactional Identity is binder.

Goals. Wil's personal goal was to create and confirm the required functionality for the application works. This is a large and consuming task, which identifies the purpose of the binding work:

Interviewer: So how do you know when your goal is done ... ?

Wil: Ah, well, when I can execute the app[lication]—well, we write tests, we'll say to test each...each unit of work like an object or a data access. So...at a certain level we know that it works—the functionality's been achieved.

Interface. In Wil's view, the designer is responsible for the look, feel, functionality, business issues, client issues, and creating user stories. Wil sees the designer as a buffer between himself and the business and client worlds. Wil says the

interaction designer “gathers the business requirements from meetings with the client and [also the] documentation from them. He gathers the requirements and fleshes that out into user stories.” Wil is happy to not be interacting directly with the client and happy to have the interaction designer solely responsible for determining the user interface and producing artefacts that he can use.

View of Artefacts Generally. For Wil, a single artefact represents a kind of high-level map that indicates the “path” someone else wants to take with their aspect of the work. These maps indicate what a client or design team member wants in terms of the product. Naturally, these artefacts play a strong role in guiding what Wil codes. Interestingly, Wil does not code from a single artefact.

In Wil’s view, new artefacts are created from others, and only some of the artefacts he “works from” are used in his binding work. Wil talks about how certain artefacts receive client approval (e.g., the stories and mock-ups), revealing his sensitivity to the stability of an artefact and his need to have stable artefacts in his binding work. Some of his views are revealed in his generalized description of the design process in his workplace:

The main stakeholder is the client. They...they provide the ideas and their vision of what they want the application to do and what it looks like. Ah, XXX gathers—XXX is the ID [Interaction Designer]—he gathers the business requirements from meetings with the client and documentation from them. He gathers the requirements and fleshes that out into user stories. Ah, from that he would build mock-ups representing those user stories. Those would be, um, approved by the client. And ah, once that’s done, um, myself (the developer) and database analyst would...would start looking at the [software] design overall, ah, [the] high-level architecture [for the code], what the application would look like, and the database schema on the database side.

As Wil sees it, new artefacts are created from old ones by “binding.” For instance, Wil saw stories and mock-ups as elements that must be bound by a binding agent, which

was the software objects that he created. In his view, the end result of binding all of the stories and mock-ups with software objects is a functional user interface. In other words, when he binds artefacts produced by others, he produces the actual user interface software and also parts of the software for the application's back end. In terms of developing the back end of the software (in this case the connection to the database), Wil says he repeats the binding process using the user stories and the database schema to create a functional interface to the database. Then, eventually, after a sequence of binding operations by himself and by others, whom he also sees as binders, the application will be complete. Although this is a fairly complex, recursive, and abstract view of work, it is not incongruous to hear such a description from a computer scientist who specializes in complexity, recursion and abstraction. In the following quote, Wil and I discuss mock-ups, stories, and database schemas in relation to the code that he must write.

Wil: Right. Ah, during the estimating of the effort, [The designer] would do a walkthrough of the mock-ups [guided by the stories] so that we understand what's required. So from that meeting we get a lot of knowledge to help us in designing it [designing is used here in the software developer's sense of the word meaning to design a high-level specification for the code or the structure of the database]. And—exactly, you're right—from the database. And XXX is there—which is a database analyst, in our case—he's at those meetings as well, to get an understanding.

Interviewer: Yeah, I can see that. 'Cause he's—

Wil: Yeah.

Interviewer: —he seems to always have his database schema in front of him—

Wil: Right.

Interviewer: And, always considering the implications to that design.

Wil: Right. Exactly. Yeah, 'cause he'll use the mock-ups, as well [as the stories], to come up with the schema. And...and on the developer's side, we do the same thing; we go with...with the schema and the user story and try to create the [software] objects that'll bind those two together.

Although many developers might claim that they work from a single artefact to write the code, Wil regularly uses two. The two artefacts he uses must be consistent with each other, but must also represent slightly different aspects of the software. A benefit of

this approach is that all aspects of the team's work tends to remain consistent with each other because binding work exposes inconsistencies. In the preceding example, both the database schema and the interface to the database were created by Wil binding stories with something else.

For Wil, visual artefacts are very important because he depends on these to do his binding work. I noticed when observing him that he communicates almost exclusively through electronic means, incorporating a range of visual artefacts and an interactive prototype/proxy of the user interface under development. Artefacts are central to Wil's work.

For Wil it is important that artefacts be stable. Wil is frustrated with meetings that are not focused or that do not directly address his issues. He does not really care which artefacts he has to bind, he just cares that someone decides what they should be and that they do not change too much. He is mostly silent in meetings. He often chooses to work at his desk in preference to participating in meetings. He and the interaction designer rarely speak, although they work together very effectively and respectfully—largely through electronic means, even though they work within 100 yards of each other.

View of Artefacts in the Project. Wil's View of Artefacts in the Project aligned with his View of Artefacts in General, which was that they are high-level maps representing the views of others and that artefacts are the elements that he must bind to produce the software. Wil likes artefacts that coordinate his work activity and allow him to remain focused. In his workplace *tickets* were used very effectively and were considered to be an extremely effective artefact by project manager, designer, developer, and database manager alike. A ticket is an item in a database that has many fields and is accessed by entering the ticket database and then entering a ticket number or writing a

query like “display all tickets for the current release of my project.” A single ticket is organized around a user story (one of the more stable artefacts on the project) and has a host of supplementary artefacts and information to help each person do their job. From Wil’s perspective the ticket artefact provided pointers to other artefacts he must bind, the environment in which he would do the binding, the tools he needed, criteria he needed to meet, and the opportunity to signal that he had completed his work. Tickets really helped organize his binding work.

View of Artefacts in Designer-Developer Interactions. From Wil’s perspective, the key artefacts used in designer-developer interactions were stories and mock-ups of the user interface. The stories were written by the user interaction designer and the mock-ups were designed by the interaction designer, the client’s agency, and an outside design company contracted to give the application its look and feel. Any time one of these parties changed their view of the mock-ups, adjustments were required, Wil’s binding work was disrupted, and his schedule slipped. When the artefacts remained stable, binding the stories and the mock-ups produced the user interface software.

Contradictions/Tensions Generally. Wil’s aim was to protect himself from contradictions. Contradictions or tensions arise in Wil’s work when the artefacts that he codes from change, creating a tension between what others require and what he has coded. Wil wants to remain productive and to focus on the work of binding. He finds that the project moves forward when team members agree on the key artefacts (e.g., stories and mock-ups). His biggest problems came when artefacts changed (which happened regularly while I was there), which meant that he had wasted effort binding/writing code that would be disposed. Wil protects himself from such wasted effort by coding the more stable parts of the application first. For instance, he understood that the mock-up was

volatile and the stories were more stable, so he relied more on stories and coded the *stories* (i.e., the behavioural aspects of the user interface) before coding the visual aspects of the user interface (i.e., the mock-up).

Wil: I don't know, sometimes, a picture is ... far easier for people to understand, makes it clearer. It's always at the high level-right?—relationship between objects, or relationships between pages, like navigation, or...or what's gonna be displayed on the page, things like that.

Interviewer: So would that be to make sure that you and the other person are on the same page about these things so that you're not wasting effort?

Wil: Yeah. It's always to—well, sometimes you wanna just brain...ah, brainstorm through it to see if it makes sense. And then, once you've es[tablished,maybe?]-once you've decided on a particular path then, yeah, it's all...everybody lines up on the same page, so...

View of Contradictions/Tensions in the Project. This category captures how Wil acted as a binder when interacting with all others on the project. Wil's View of Contradictions/Tensions in the Project aligns with his View of Contradictions/Tensions in General, which is that they should be avoided. The Contradictions/Tensions in the Project concept contained many quotes on changing requirements. Requirements changed because of client requests and the continually evolving context for the application. The other issue was the target delivery date, which was tracked by the project manager. Wil managed his binding work within the constraints of the target delivery date and the volatile requirements by working on his project manager's priorities.

View of Contradictions/Tensions in Designer-Developer Interactions. When interacting with the designer, Wil understood the pressures the interaction designer was under. Although changes did happen to the mock-ups, Wil was careful to protect himself by coding the more stable parts of the application while he waited for the interaction designer to settle on a design. Wil worked very well with the interaction designer and understood the designer's task. In fact, Wil and the interaction designer were unique in

the depth of understanding they displayed for each other's work. Their backgrounds helped here. The Interaction Designer had worked as a developer previously and the developer had had some training in graphic design.

Thus, Wil's case is typical in that all of the eight coded categories together indicated one singular identity. Figure 9 indicates that all eight categories support the view that Wil sees himself as a binder. It is an interesting identity, because the original two artefacts that are bound by Wil cease to be relevant after the binding operation, except that they exist indirectly and invisibly through the code that Wil has written to reflect these artefacts. Wil's case is also unique in that no other developer presented this particular Interactional Identity.

Figure 9 indicates the two influences on the Interactional Identity that were observed in this study. The first is education. Wil's background is in software and hardware and includes some graphic design courses; however, it is unclear how this might be related to Wil's Interactional Identity as binder. It is possible that the connection with the concept of binding comes from his software courses, a high school chemistry class, or just general knowledge. The second connection is between project circumstances and Interactional Identity. Wil is not a team leader and his role as binder fits well with someone who must accommodate himself to the work of others. This is especially true because Wil and his team were engaged in contract work.

Figure 9. A model of developer Wil's Interactional Identity, binder.

Abbreviated Case Studies

Other psychologists, analyzing in the same way, may produce somewhat different phrases or handles to describe a person's Interactional Identity. However, I would expect another psychologist to be able to confirm the model of Interactional Identity presented in the previous section. My confidence in this statement comes from analyzing the data and identifying the Interactional Identities of 18 additional participants (I did not analyze one developer's transcript, because I had reached saturation in the category of more experienced developers). In all cases, the concepts and categories named and identified here could be used to determine an individual's interactional identity, one that was consistently reflected in all concepts and categories. These analyses can be found in Appendix F, where each participant's Interactional Identity is briefly described. Table 16 displays the Interactional Identities of 20 participants, as well as their years of experience.

Table 16 *Interactional Identities Grouped by Specialist and Experience*

Experience	Designers		Developers	
	Interactional Identity	Years of experience	Interactional Identity	Years of experience
Less	Incorporator	2	Assembler/builder	3
	Pioneer	2	Catalyst	7
	Negotiator	6	Problem solver	10
	Closer	8	Practitioner	10
	Reformer	10		
More	Movie director	10.5	Historical anomaly	15
	Quality advocate	11	Training advocate	17
	Steward	12	Pragmatist	20
	Facilitator	13	Binder	22
	Teacher	33		
	Artist in business	40		
	Elder	40+		

Table 17 *Interactional Identities—Handles and Descriptive Phrases*

Identity handle	Descriptive phrase
Designers	
Incorporator	Incorporates their design work with the work of others
Pioneer	Opens up new work in the field of user needs analysis
Negotiator	Negotiates with others to arrive at design decisions
Closer	Closes in on a design ‘deal’ that is agreeable to all stakeholders, including himself
Reformer	Reforms organizational structure to achieve greater compliance to design standards
<hr/>	
Movie director	Acts as a director of a cast and crew in as in rehearsal mode for production of a movie
Quality advocate	Advocates for product quality
Steward	Acts as the team’s steward for producing useful software for his product’s target audience
Facilitator	Acts as a facilitator of a distributed, problem-solving activity
Teacher	Engages others in courseware production through teaching
Artist-in-business	Uses his art to achieve business goals for the team
Elder	Has seen software-creation work evolving over time
<hr/>	
Developers	
Assembler/builder	Assembles parts from others and produces parts of his own to build a product
Catalyst	Acts as a catalyst in the team to ensure changes in the product are made so that the product is usable
Problem solver	Can find satisfactory solutions to perplexing and difficult software problems
Practitioner	Reliably and professionally delivers and expects a high standard of work in the team
<hr/>	
Historical anomaly	Holds a unique position in software work and is misunderstood. Must constantly explain role to others.
Training advocate	Advocates training concepts and uses training techniques to deliver this message to the team
Pragmatist	Aims to help the team establish and meet achievable software goals
Binder	One whose aim is to develop software that binds together the work of others and creates something that is functional.

Note. The dash lined divides the specialists with more and less experience.

The Interactional Identities Model

In the last section I argued that each participant has a unique Interactional Identity, one that colours designer-developer interactions and the meaning that artefacts have for individuals. The variety of Interactional Identities adopted within each of the two groups was a clear result, as no two individuals within either group had exactly the same Interactional Identity. An outcome of the previous analysis was the construction of a model for an Interactional Identity, which comprised elements that were deemed to constitute/reveal it and that are captured by the concept Interactional Identity.

This section scrutinizes the Interactional Identity model in more detail in order to discern what it reveals about designers and developers' talk- and artefact-mediated interactions. It focuses on the concept of Interactional Identity and reconsiders the issue of the specialist's approach to joint work and what artefacts mean to them in this context. However, unlike earlier sections, this section considers designers and then developers as a group, rather than as individuals. Figure 10 depicts a general model of Interactional Identities that was developed in the previous sections to capture a designer's or developer's unique approach toward their joint interactions.

Figure 10. The Interactional Identity model.

Actual Work and the Interactional Identity

Interactional Identity was a construct created as the result of a detailed analysis of interviews with 20 individuals that covered a range of topics related to the activity of creating software. These interviews emphasized the topic of artefact use. Eight of these individuals were also shadowed in the workplace prior to their interviews and in all eight cases there appeared to be little or no difference between how an individual said they worked and how they actually work.

However, this is not to say that it is easy to discern an Interactional Identity by observing behaviour at work. The Interactional Identity emerges more clearly in interviews, where the participant has the opportunity to *explain* their behaviour (i.e., their rationale/perspective). The combination of the two methods provides the opportunity to confirm the Interactional Identity of the individual and understand the meaning that individual ascribes to artefacts.

Discipline Specified Roles and Interactional Identities

The Interactional Identity of designers and developers in the workplace appears to bear little semblance to discipline-specified roles (e.g., “developers are like engineers”). The Interactional Identities observed among the eight developers were assembler/builder, catalyst, problem solver, practitioner, historical anomaly, training advocate, pragmatist, and binder. None of the developers in this study had Interactional Identities that suggested they saw themselves as an architect or any type of engineer and none of them saw themselves as a member of a well-defined profession. The Interactional Identities of six developers had a practical bent and were more like a trade person’s identity. The

assembler/builder, problem solver, training advocate, pragmatist and binder are all Interactional Identities that suggest concrete tasks.

The following Interactional Identities were observed among the designers: incorporator, pioneer, negotiator, closer, movie director, reformer, quality advocate, steward, facilitator, teacher, artist-in-business, and elder. The historical review showed that at the discipline level designers describe themselves in terms of the nature of their role (e.g. it is expansive by nature), rather than by analogy to other professional roles. One issue for designers was whether or not their role was specifiable. Four designers appeared very certain about how to design (either in general or for the specific type of software they produced). These were the movie director, teacher, facilitator, and the pioneer. Another issue was whether or not the designer's role was diffuse and expanding. Two people clearly saw their role dynamically. The elder repeatedly emphasized the link between changing technologies and the changing role of the designer, and the facilitator emphasized his lifelong commitment to learning. With regard to whether or not designers saw themselves as professionals, no one had a view of him or herself as a professional in a well-defined discipline; however, four designers stated they worked to published standards in their domain. These were the elder, teacher, closer, and steward.

This analysis exposes a large gap between discipline-specified roles such as engineer and architect and the actual Interactional Identities for the developers in the workplace. There is also a significant gap between the Interactional identities of designers and the nature of the role as expressed at the discipline level.

Roles as Understood in CHAT and Interactional Identities

In cultural-historical psychology, a role is an identifiable portion of a larger task that is the responsibility of an individual. In this case, the larger task is constructing a new software product. Each role has a different responsibility in the group's activity and the various roles are relatively clear. Job titles are indicative of a person's role.

An Interactional Identity is not a role as understood in cultural-historical activity theory. The Interactional Identity is more an expression of an individual's personality or a summary of their behaviour in a specific activity. It is how that individual brings their role to life and gives it a unique interpretation. A role is more closely tied to tasks. An Interactional Identity is the approach an individual takes to a task.

Individuals and Interactional Identities

Designers and developers do not appear to be consciously aware of their own or others' Interactional Identities just as people in general are not consciously aware of their personalities. The Interactional Identities were discerned by a variety of indicators in the transcribed interviews, ranging from the condensed spreadsheet data, quotes which revealed a participant's choice of words, repeated phrases, and the overall impression of the interviewee. Exceptions would include those who explained their approach to their work by analogy and identified their Interactional Identity by telling me directly. This would include the movie director "It's very similar to what a movie director has to do" (Owen); the steward "But at the end of the day we are the stewards, right?" (Mason); and indirectly, the problem solver "We are involved in the framework, which is problem solving" (Damian). By inference this also means that designers and developers are likely not consciously aware of the unique way that they use artefacts in their interactions with

each other. However, not being conscious of one's Interactional Identity does not mean that one should not be able to recognize one's Interactional Identity or another person's Interactional Identity if it is described.

The Relationship Between the Designer and Developer's Interactional Identities

A designer-developer pair working on the same project need not and, in fact, did not possess "matching" Interactional Identities. For instance, the developer who was a problem solver did not need to be paired with a designer who posed problems in order for interactions to be functional and productive. In fact, the developer with a problem-solving Interactional Identity was just as likely to be matched to a designer who was an incorporator as he or she was to the steward or to the closer because there were no patterns whatsoever to the pairings.

This mismatch was not a problem in interdisciplinary interactions and may have even aided the interactions because mismatches ensured that more ground was covered in a meeting. A designer who perhaps was a steward and focused on his or her responsibilities to end users might be nicely balanced by a developer with a different approach, say a pragmatist, who would be most interested in creating a feasible design. Together the two would be placed in the position of resolving the contradiction created by their disparate Interactional Identities and the ideal consequence for the project, in this specific example could be an "implementable" and implemented design solution that would benefit end users. Similar benefits could also be extrapolated from any nonhomogenous pairings of designers and developers.

Thus, in joint interactions, others regularly reinterpret (not misinterpret) artefacts on the fly depending on their Interactional Identities, suggesting that meetings of

individuals with disparate Interactional Identities and the reinterpretation of artefacts by developers and designers creates necessary contradictions or tensions that, when resolved, benefit the design work by progressing the design so that it is satisfying to two or more individuals with nonhomogenous Interactional Identities.

Projects and Interactional Identities

Given that this research focused on the early design phases of a project, it is difficult to say whether a person's Interactional Identity persists throughout other stages of the project. However, some observational evidence suggests that this is the case. For example, the steward appeared to always be a steward regardless of what stage of a project he worked on (recall that the observation sessions, although focused on one particular project, inevitably resulted in observations of work on other projects at different stages because designers typically worked on several projects simultaneously). However, developers generally worked on one project at a time, so similar inferences cannot be made regarding developers.

Organizations and Interactional Identities

Each participant was observed in only one organization so it is difficult to determine if Interactional Identities change when an individual changes organizations. However, it can be said that organizations appear to afford or constrain designer's Interactional Identities. Without going into details, the issue here is the organization's culture. For example, the following Interactional Identity-organizational culture pairings are not surprising: the movie director designer was part of a small, creative and dynamic organization. The closer was part of a large, less innovative, and more rigid organization, and he was therefore very oriented to moving the work forward. The steward was part of

a service-oriented organization. The negotiator was part of an organization that had hired a team of contractors that was working to tight deadlines. The artist-in-business was part of a tiny start-up where survival of the company was an ongoing concern incorporated into the artist's role.

Developers' Interactional Identities appear to be less tied to the organization, with the exception of the training advocate whose identity was tied to the department's purpose.

That workplace culture appeared to affect the Interactional Identities of designers, but not developers, may have been due to the developer's role being better established than the designer's role. Organizations therefore create new opportunities for designers "to be," but at the same time constrain what they can be. By contrast, the role of the developer is well understood, there are many positions for developers, and their role is generally not constrained in the sense that their role meets their expectations.

These results suggest that designers' Interactional Identities may shift or be transformed more when designers change organizations, whereas developers' Interactional Identities may remain more stable.

Team Work and Interactional Identities

The data for the various Interactional Identities were derived primarily from interviews that focused on the topic of designer-developer interactions. However, there was a strong connection between participants' Interactional Identities and their actual interactions with their teams. This was confirmed by the data from participants who were both interviewed and observed (7 of 12 designers and 5 of 9 developers). In almost all instances when designers and developers interacted in a meeting, there was also at least

one other person present who represented the business side of the organization.

Frequently, other team members were also present. The most attendees observed at one meeting was 11, but 4 or 5 was typical. In these circumstances, individuals, such as the designer who was the closer, approached all team members on the project from this same standpoint, including managers who represented the business perspective, project owners, testers, and documentation writers. This was also true of developers; their Interactional Identities were consistent across all their interactions with other team members.

Hence, the Interactional Identity represents an approach to interacting that is used in all interactions, including artefact-mediated interactions, with other team members. Conversely, these data show that in this regard, the interactions between designers and developers are similar to the interactions a designer or developer would have with any other team member. This implies the Interactional Identity is tied to the activity.

Interactional Identities Over Time

Interactional Identities and Their Development Over a Career

A little can be said about the development of Interactional Identities over the course of a career. Table 16 groups Interactional Identities by specialist type and years of experience in the discipline.

Designers with less than approximately 10 years of experience all developed Interactional Identities that expressed their assumption that they controlled the design of the user interface and their Interactional Identity appears to reflect the frustration they feel as a result of being unable to control the design the user interface. The incorporator carefully adapts her contribution to others around her (including the developer); the pioneer finds subversive and creative ways to reach her end users to determine the right

software to develop and to ensure that it is developed; the negotiator liaises with many team members (including the developer) to come to agreeable solutions; the closer creates opportunities where conflicts between the developers and management are exposed and resolved; and the reformer advocates for creating a new managerial position among the developers so she can better control the user interface. While reflecting the fact that inexperienced designers are focused on the task of designing the user interface, these Interactional Identities also capture the designers' frustration originating from their lack of control over the design of the user interface on their respective projects.

By comparison, more experienced designers expressed less frustration and a more positive approach to joint design and development work, even though they too were not fully in control of the design of the user interface. For instance, the movie director fosters creative activity within his team to progress the design work of the group; the quality advocate accepts that his role is to push for the highest possible quality product for end users, accepting he will not achieve this end. The steward sees it as his duty to consistently take the perspective of the end user; the facilitator of distributed problem-solving activity relies on the input of others to achieve an optimal design; the teacher is open to suggestions from all team members, the artist-in-business keeps short-term artistic issues and long-term business issues in mind when designing; and the elder understands the project context and the possibilities for user interface design historically.

Designers, then, initially adopt an Interactional Identity that allows them to cope with the powerlessness they experience in their projects and organizations in terms of executing their perceived task, which is user interface design. Later, as they become more experienced, designers appear to craft new efficacious and more positive Interactional

Identities that allow them to not only work effectively within this constraint, but to understand it and see the advantages. Figure 11 illustrates the broad differences in Interactional Identities for more- and less-experienced designers and developers.

Figure 11. A Model of designers' Interactional Identities based on years of experience.

Figure 12. A Model of developers' Interactional Identities based on years of experience.

The developers who are less experienced present themselves as focused on the task of writing software, the difference between the less-experienced developers being largely in how they accomplish this task. The assembler/builder constructs code; the catalyst causes the necessary changes in the design so that the right code will be produced; the problem solver figures out what code must be written; and the practitioner skilfully manages the code production process by producing careful estimates and ensuring he understands exactly what software is required.

By comparison, more-experienced developers present themselves as more oriented toward the social aspects of development work. For instance, the historical anomaly developer sees his interactions in terms of his experiences on previous projects; the dealer and salesperson of training concepts sees himself as a person engaged in reorienting his entire department; the pragmatist sees himself as the person who grounds the business person and designer; and finally, the binder sees his work in terms of the work products that others produce. As illustrated in Figure 12, then, each of these developers sees themselves not so much in terms of their task, but in terms of their relationship with others on the team, including the designer.

This suggests that roughly speaking, in the first decade of their career, developers are focused on code production, in contrast to their later years when their objective widens as reflected in a transformed and expanded Interactional Identity.

For both designers and developers, Interactional Identity appears to develop, albeit differently, over the course of careers. The pattern seems to be a broadening of focus for developers (from task to team issues), while for designers the pattern indicates an initial adaptation of the Interactional Identity to address the lack of control that

designers have over the design of the user interface to the development of a Interactional Identity that is more efficacious and acknowledges and incorporates the role of other team members in the design of the user interface. In both cases there is a broadening of Interactional Identities.

In general, Interactional Identities appear to develop slowly over the course of a career in terms of their nature. As a result, the meaning that designers and developers ascribe to artefacts also evolves slowly over time. Both designers and developers tend to see artefacts more and more as tools for collaboration as they garner experience in their roles, and their Interactional Identities and interactions with others reflect these changes.

Development of the Key Components of an Interactional Identity Over Time

Several critical categories or components comprise the Interactional Identity. The data used here are derived from a spreadsheet that summarized each individual's orientation on the 15 different concepts that emerged from the coding process. However, only four of the eight concepts that play a role in capturing Interactional Identity are reported here. An exploration of the distinctions between role and experience is carried forward in this section to show that, as expected, these concepts differ for those with less than approximately 10 years of experience relative to those with more than approximately 10 years of experience. This is because the higher-level category reflects all the data at the lower-level concepts. Subsequent sections confirm this.

Goals. To understand individuals at work, this study enquired about personal goals. Less-experienced designers were focused on the goal of producing the user interface design required for their project. They have task-focused goals. In contrast, more experienced designers are not solely focused on producing a design. They are more

aware of the value of the product to end users and their organizations, as well as the various attributes the software should possess. They also think ahead to future products that may be built on the current product or tools that might be created as a by-product of creating the current product. Table 18 summarizes designer goals. Each item in this table and all following tables in this section reflects the position of a single individual. The participant with the very least experience in each subgroup is presented first in the column and the one with the most experience is presented last.

Table 18 *More- and Less-experienced Designers' Goals*

Less-experienced designers	More-experienced designers
To ensure teaching goals of the game are met by aligning all elements especially teaching and gaming. (Incorporator)	To create a framework and a gaming method (the generalized form for the game). (Movie director)
To design an effective way of filling out the search form that fits the context of how users execute and re-execute searches in the context of their work over long periods of time. (Pioneer)	To develop the HTML code for the application UI, determine the look and feel, ensure aesthetic quality.(Quality advocate)
To design the screens in the new look and feel, incorporating business changes into the workflow, while retaining all other aspects of the old system. (Negotiator)	To rescue the project and ensure that the design meets a host of objectives: implementable, satisfies Web standards, has the right functionality, is responsive, intuitive, accessible. (Steward)
To produce a design that management and R&D would "buy into." (Closer)	To benefit end users through his design work and to learn about and develop tools for design work. (Facilitator)
	To produce something that's really useful to end users. (Teacher)
	To design a productive user experience. (Artist-in-business)

Less-experienced developers wish to complete their task. They are product-focused. The exception was the most experienced of the less-experienced developers who

had 10 years of experience and who was also process- and project/business-focused. More-experienced developers tend to be more project- than product-focused. They not only want to create the product, they want the code to be reusable and they want it to meet high standards. An exception was the most experienced developer whose stated goal was to create the required functionality. Table 19 lists the goals of the more- and less-experienced designers.

Table 19 *More- and Less-experienced Developers' Goals*

Less-experienced developers	More-experienced developers
To contribute to the design and to produce the game. (Assembler/builder)	To bring the designer's designs to life, provide pushback on the designs, produce reusable XAML code. (Historical anomaly)
To complete the feature in a timely manner, to produce something that's usable and consistent with the feel of the product, to fulfil the work order and satisfy the customer. (Catalyst)	To design the best, up-to-date, efficient product possible and to break a "bad habit" of the design team, which is conducting a learning needs analysis, rather than a training needs analysis. (Training advocate)
To create a framework for GUI development, to reproduce the previous functionality in a Web environment. (Problem solver)	To leverage the creativity of others to produce an application that performs well and provides the functionality determined by others. (Pragmatist)
To build the application specified by the designer, to improve processes by building tools, to build maintainable software, to create a successful product, to mitigate project risks. (Practitioner)	To create the required functionality. (Binder)

Interface. The historical review revealed a potential conflict over ownership of the user interface, so this was explored in the interviews. The goal of the less-experienced designers is to propose a successful design (or their part of the design) because they perceive this as their responsibility (i.e., they own that part of the project work). The less-experienced designers' frustrations and compromises are reflected in the summarized

data shown in Table 20. By comparison, many of the more-experienced designers, while taking responsibility for the user interface, regard themselves as supporting the process of designing the interface, rather than being solely responsible for it. In various ways, they actively collect the views of other team members, balance them against each other and known design principles, and reproduce a new design proposal. More-experienced designers struggled with their ego (see Table 20) and the assumption that they must produce the design. For more-experienced designers, the focus is much more on the design emerging through team interactions. Table 20 lists the perspective of experienced designers on who determines the user interface.

Table 20 *More- and Less-experienced Designers' Views of the User Interface*

Less-experienced designers	More-experienced designers
A successful design is one where the work of all the designers is successfully integrated. This is a fun process that involves interacting with designer peers and also the developer who also acts as a designer. (Incorporator)	Struggles with ego so that the ideas of others are not disregarded. (Movie director)
The user interface can't be designed properly without access to clients and end users (i.e., the user interface design is the designer's responsibility). (Pioneer)	Checks ego at the door. (Quality advocate)
The designer owns the user interface, but they must please a host of others: business managers, other designers, developers, testers etc. (Negotiator)	Sees self as an egoless designer. (Steward)
Designs reflect what is good design and what the managers want, and what the developers need. (Closer)	Owns the design, but brings a lot together or ties a lot together when designing. Considers the viewpoints of the user, the engineers and business. The reconciliation of these viewpoints is the problem that must be solved. (Facilitator)
Designers are responsible for the design and adhering to Web and accessibility standards. (Reformer)	Feels responsible for the design, but is very open to help from others and feedback from auditors. Sees self as a learner too. (Teacher)
	First priority is that the business requirements are met. The second is that the design can grow. The third is that the design must be workable. Designing is both a collaborative and a private activity. (Artist in business)
	The designers create the designs, confirm them with the end users, and work with the technical people who turn it into working code. (Elder)

In contrast to less-experienced designers, less-experienced developers are of the opinion that designers are not solely responsible for the design of the user interface, a likely source of frustration for less-experienced designers. For experienced developers there is an understanding that the designer is responsible for the user interface, but also an

understanding of how the user interface emerges through interactions with others on the design team including themselves. These views are reflected in Table 21.

Table 21 *More- and Less-experienced Developers' Views of the User Interface*

Less-experienced developers	More-experienced developers
There is shared responsibility for the design, particularly in the beginning stages where everyone is a designer. (Assembler/builder)	The designer specifies the UI through interactions with the analysts and the subject matter expert, but the designer/developer and the developers offer opinions. (Historical anomaly)
The designer produces the initial proposal and the developer and other stakeholders shape and alter it (sometimes significantly) with their responses. (Catalyst)	Designers determine the overall structure of the course and the content. The developers settle on visuals (images), interactional elements (buttons, input boxes), and flow of control within a page of the course. (Training advocate)
The designer is responsible for designing the UI and he is a part of the team. The design is not “interdependent with” (i.e., it is dependent on) the work of the developers. (Problem solver)	The designer and the president of the startup mainly work out the design for the user interface between them. The president has his ear to the marketplace, clients and end users. The designer suggests concepts, ideas and is generally creative. (Pragmatist)
Ultimately the developers and designers both answer to the product team—the financial analysts, who hold the real power. (Practitioner)	The designer is clearly responsible for the look, feel, functionality, business issues, client issues, and user stories. The designer is like a buffer between the developer and the business and client’s world. (Binder)

These data suggest that both developers and designers develop a more nuanced view of who determines the user interface over the course of their careers, resolving initial views that are in conflict with each other and possibly also in conflict with project goals. These data, gathered from functional pairs of designers and developers, indicate a significant discrepancy over the issue of who determines the user interface that could be a source of tension on projects where there are less-experienced designers or developers.

Artefacts. Artefact use is a key concern of this dissertation. All of the developers and designers viewed artefacts as essential to their work. Observations confirmed that artefacts are ubiquitous in the work environment and are present at every meeting between designers and developers, except for perhaps the most casual exchanges.

For less-experienced designers, artefacts are regarded as things that support the subtasks of the task of designing a user interface, such as supporting the task of communicating a design, exploring a proposed design, helping to sign off on a design, or managing the work of designing. For experienced designers, artefacts help to discern or develop the user interface in the context of the work of the entire team. Table 22 summarizes the purpose of artefacts according to the views of more- and less-experienced designers.

Table 22 *More- and Less-experienced Designers' Views of Artefacts*

Less-experienced designers	More-experienced designers
To support communication: this is what I want, this is what exists. (Incorporator)	To support team work. (Movie director)
To explore. (Pioneer)	To support a negotiation process, to determine what will be developed to what standards and when. (Quality advocate)
To expose what one wants or thinks, to guide the implementation of the UI, to ask questions, to help the developer think through their issues. (Negotiator)	To help take charge of the situation and progress the work. (Steward)
To support specific tasks to be performed that settle issues or reach agreements. (Closer)	For group problem solving. (Facilitator)
To manage the work. (Reformer)	To support the creation of training material. (Teacher)
	For testing ideas with others, communicating, raising problems, exploring the common ground. (Artist in business)
	To turn normal people into experts. (Elder)

For less-experienced developers artefacts are seen as aids to progressing the work.

For experienced developers, artefacts are aids to communication, aids for group problem solving or objects that represent the work or the position of another person on the team.

These views are summarized in Table 23.

Table 23 *More- and Less-experienced Developers' Views of Artefacts*

Less-experienced developers	More-experienced developers
“Every level of design has a different demand on what artefact can be used” meaning that interaction stories are a good starting point, but addressing arrangement issues requires a sketch, and a prototype is required to get at the interaction (i.e., this is about results). Artefacts help the team to complete work. (Assembler/builder)	They are used for communication purposes. (Historical anomaly)
Artefacts are a catalyst for change. (Catalyst)	Artefacts support the communication task. (Training advocate)
Artefacts help with the task of building. (Problem solver)	They are a means to an end. They are primarily cognitive aids (individual or group) for problem solving. (Pragmatist)
Using artefacts well is a discipline that this developer enforces in his projects to combat folklore (meaning assumptions) and chaos (conflicting assumptions) and ultimately to get better outcomes. (Practitioner)	“Artefacts are binding agents.” Two artefacts binding together result in the production of a third artefact, which itself can be bound to another artefact, which will ultimately create code (also an artefact). High-level artefacts of each stakeholder must fit together so each person can work on his or her own. (Binder)

Contradictions/Tensions. Less-experienced designers experience tensions on the project as problems, whether the tensions are set up deliberately or are just “everywhere.” Experienced designers see tensions more positively (e.g., as natural, as problems to be solved within the group or within a broader context, or as opportunities) as shown in Table 24.

Table 24. *More- and Less-experienced Designers' Views of Contradictions*

Less-experienced designers	More-experienced designers
The company sets up contradictions deliberately because they help with game development. (Incorporator)	They're learning opportunities. (Movie director)
Tensions should be resolved directly and not avoided. (Pioneer)	Sometimes there's just too much compromising, especially with quality. All parties should be involved in resolving contradictions. (Quality advocate)
Tensions should be resolved by compromises. Design is about tensions and negotiations. (Negotiator)	Tensions can be solved by heroic individuals, by people getting together and talking openly and using common sense. (Steward)
Tensions pull the project in different directions instead of just one direction. (Closer)	They're part of the occupation. You resolve them by talking to people, creating and presenting the right artefact and developing one's toolkit. (Facilitator)
There are problems everywhere and plenty of room for improvement. (Reformer)	Tensions resolve contradictions by identifying something that is "possible to achieve" within the team and that is "meaningful" to end users. (Teacher)
	Tensions are addressed by keeping eyes firmly fixed on real business objectives and not sweating the small stuff. (Artist-in-business)
	A historical perspective helps in understanding the evolutionary pressures/forces/tensions on the project at hand such as: the general appreciation in the public for what computers can do increasing over time, the skill set of the team increasing over time, and increasing role of technology. All these changes influence what can be developed today as opposed to previously. (Elder)

Less-experienced developers view contradictions (i.e., tensions) as issues or problems, something to be accepted, or something to be controlled. Their attitude to tensions is negative or neutral. Experienced developers learn to live with contradictions

or tensions on the project. They are aware of them, resolve them, work around them, see them as mere roadblocks, or see them as a good thing. Experienced developers are empowered faced with contradictions. Views of tensions are shown in Table 25. In general, contradictions are viewed as personal problems by less-experienced individuals and as motivators to act by those with more experience.

Table 25 *More- and Less-experienced Developers' Views of Contradictions*

Less-experienced developers	More-experienced developers
Tensions are natural. (Assembler/builder)	It's important to be very aware of a wide range of project and activity-related tensions. Tensions that can be resolved are addressed immediately through face-to-face chats. (Historical anomaly)
Tensions are issues to be addressed. (Catalyst)	Contradictions are merely roadblocks to achieving [the] goal of selling important training concepts and techniques. (Training advocate)
Tensions are problems to be solved. (Problem solver)	Conflict is a good thing because it pushes the envelope. (Pragmatist)
Control the process and be disciplined about the process because those are things that can be controlled. Other issues that are not controllable are accepted, such as how hard the problem is or unexpected issues with implementation. (Practitioner)	The important thing is to protect one's self from contradictions, especially rapidly changing requirements by coding intelligently. It's most important to stay focused on the work of binding and not get involved with noncoding issues. (Binder)

Outside the Interactional Identity Model

Seven coded concepts were not used to determine the Interactional Identity because they bore no correlation to it. Thus, not all coded categories revealed an Interactional Identity. Two that do not reveal Interactional Identity are discussed here. The first, which is particularly surprising, was the individual's View of the Other Practitioner. The second was Project Goals.

View of the other practitioner. In general developers viewed designers as individuals who are focused on the concrete aspects of the production of the user interface, such as the appearance and organization of the visual aspects of a screen layout. At times they did not understand the complexity of the designer's role. Only the last developer whose View of Designers is shown in Table 26 expressed a fully developed view of the scope and complexity of the designer's role.

Conversely, many designers viewed developers negatively, suggesting they have to be controlled, they do not communicate well, or they are strange. Alternatively, designers viewed developers in stereotypical ways, voicing opinions such as, "developers do not have a broad perspective," "their world is black and white," or "they understand complex technical stuff." However, a few designers had very positive views, such as the teacher who said developers create, are helpful, and check work. Another positive example was the elder who saw developers as collaborators and "more normal" than the developers of previous eras. Designers' views of developers are summarized in Table 27.

Table 26 *Developers' Views of Designers*

Developer	View of designers
Assembler / builder	Each designer is unique, but they all communicate with developers through artefacts. They each have unique interests and do different things. Each designer looks at a different aspect of the product.
Catalyst	Very unsure about the different types of designers and their titles "however you want to call it" (Catalyst, 209). Says concerns are look, feel, usefulness and usability. Not sure what a design review is and is not clear on the design process for user interfaces. The developer sees the designers as document-centric.
Problem solver	The designer's work is simple.
Practitioner	Designers decide colours, arrangements, the text, the elements and the workflow. Designers provide valuable input to the core work, which is done by the developers and the product team. The developer is sympathetic to their lack of power in the organization, but until that changes, the developer answers to the product team.
Historical anomaly	Believes the analysts determine the how and why of functions and the designers create graphical user interface designs. The developer thinks the designers main concern as creating something that is friendly, cool, slick and professional. The developer does not see the lead designer as a designer, but rather as a subject matter expert.
Training advocate	Designers determine the structure of the course. The developers determine the visual and interactional elements (like button look and feel, logos, ...) as well as the flow of the interaction within a page of the course (e.g., within a page of multiple choice questions).
Pragmatist	The designer is an Artist with ideas about screen design, UI concepts, usability, and flow of interaction.
Binder	Has a very clear understanding of the designer's role and work. Emphasizes the transformative aspects of the designer's work. Sees the designer's work in terms of input, output and transformations.

Table 27 *Designers' Views of Developers*

Designer	View of developers
Incorporator	Developers care about functionality and logic i.e., the behaviour of the user interface. The designer is dependent on the developer for the interactive, realistic prototype. The developer can also be a designer because he is a gamer.
Pioneer	Developers do whatever they want. UI is not a priority for them.
Negotiator	The UI developer faithfully reproduces the designers' design. The relationship is very good.
Closer	Developers want interesting, but easy work, they are not innovative as a rule, not prepared for meetings, they're builders, they make it real, sometimes they do whatever they want.
Reformer	Developers are people who need to be controlled so that they follow standard processes and code to standards.
Movie director	Developers are not great communicators by training, they do not necessarily have opinions about other aspects of the product, they compartmentalize themselves.
Quality advocate	Developers understand complex technical stuff, what they do is hard, they care about quality (bug free code) and functionality and sometimes cost, their job is to deliver as quickly as they can, so they do not create extra work for themselves. Their world is black and white.
Steward	Developers want to complete as fast as they can, they build the product, process tickets, take a short term view, they do what the designers tell them to do, they're professionals, they have good ideas, do a lot of work.
Facilitator	Developers are experts who can implement features, help the designer to understand how big a solution is or how difficult it is.
Teacher	Developers have a different mandate They create and help. Designers and developers each do their part and also check each other's work.
Artist-in-business	Developers are knowledgeable, strange people, they deal with the new alchemy. The user experience is the common ground between the developers, designers, and business person.
Elder	Developers are collaborators with the designers and work together comfortably. Developers are now more normal people in comparison to previous times when there were no tools, practices or processes. Designers and developers are now closer to their technologically savvy clients and end users.

Project Goals. Stated project goals do not contribute significantly to understanding the Interactional Identity, as these mainly reflect business objectives for both designers and developers. For example, one developer said the project goal was to modernize the product, ease future development and deployment (to reduce costs), and produce a workable GUI for users. One designer said the project goal was to satisfy the client. The goals reflected business, client, or end-user goals.

Concluding Statements

This analysis has shown that the interactions of each designer and developer are coloured by his or her unique Interactional Identity. This identity is reflected in how the designer or developer views their work, including how they view contradictions, their view of who determines the user interface, how they see the role of artefacts in their work, and how they interact with their fellow specialists through artefacts.

In practice, Interactional Identities seem to serve the purpose of organizing the individual's work and making the work meaningful to the individual. The Interactional Identity is about how to get the work done in collaboration with others. For instance, the movie director designer treats his team as members of a cast and sees artefacts as important for coordinating everyone, but secondary to the drama of developing a game. The negotiator designer sees herself as a person who negotiates a design that is agreeable to multiple stakeholders in her organization and valuable to end users. The catalyst developer works by responding to designs to point out gaps and deficiencies that need to be changed. The pragmatist developer is firmly focused on the end goal of producing the

required software and sees artefacts as aids to achieving this end and other team members as fellow collaborators.

There is significant evidence to suggest that Interactional Identities are tied to an individual's work situation and may be an adaptation to that situation, although the research did not follow the individuals long enough to verify this. The historical anomaly is a good case in point, as his identity is an adaptation to his odd role within his organization. He uses his personal skills to appease designers on one side (he makes picture-perfect reproductions for the designer and is careful not to do her design work for her), and developers on the other (he follows the guidelines that specify exactly what he is and is not responsible for with respect to coding to the letter).

The study's findings show that designers and developers adopt Interactional Identities. Each designer and developer chose to present themselves as being in a certain type of relationship with other team members. This identity was revealed both directly and indirectly throughout the entire interview. Table 17 summarized the Interactional Identities that were found.

The analysis shows that artefacts have another level of meaning to practitioners beyond their obvious function of representing a plan or idea. Someone with a certain Interactional Identity will select and use artefacts differently than someone with a different approach to interacting. Further, the whole meaning of artefacts for the individual is coloured by their Interactional Identity. The teacher designer will be using them to teach; the closer designer will be assessing an artefact's value for finalizing agreement on a design; the assembler/builder developer will be assessing an artefact for

its value as a part; the binder developer will be asking if the artefact is stable enough to code from, etc.

The Interactional Identity is significant not only because it reveals how artefacts are used, but because it uncovers a deeper level of motivation for individuals engaged in an activity—one that links the individual, their background, and the situation in that individual's workplace or organization. The designer and developer are no longer just seen as motivated to design the user interface and develop the software. They are also individuals who must negotiate a complex culture. The steward feels his need to advocate for end users in his design work because of the type of organization he works for. His level of frustration with his client is huge because she makes it hard for him to be a good steward. The designer who is a *pioneer* has become an advocate for change in how software is designed precisely because her organization still has a software development- and profit-focused culture. Similarly, the practitioner has the luxury of being a professional developer because his organization values developers more. And finally, the historical anomaly feels like a bit of a misfit and spends most of his time in face-to-face talks with “real” developers because the “real” developers in his organization as a group disrespect his particular role as a designer/developer.

Because motivation is linked closely to need, Interactional Identity also reflects human needs, both personal and organizational. The movie director may fulfill the needs of his personality with his identity by strengthening its ego, but the movie director's identity also fulfills a need within the organization to coordinate game development work. Similarly, the binder may fulfill the needs of his introverted personality by

allowing him to focus on artefacts instead of people, while also filling a need within his organization to produce the client's software.

Thus, the Interactional Identity does not just reflect how designers and developers interact through artefacts, it also captures aspects of the individual and the project or organizational context because it captures how a designer or developer works with others on the project team, as well as capturing individual and organizational motivation and needs.

Interactional Identity and the Meaning of Artefacts

This section has shown that the Interactional Identity, which is tied up with how a designer or developer ascribes meaning to artefacts, has certain properties. Firstly, the Interactional Identity, which is best discovered through the interview process where designers and developers can explain their approach, is closely associated with how a designer or developer actually works, including their use of artefacts. However, it is not closely associated with discipline-specified roles, particularly in the case of developers, who did not see themselves as architects or engineers.

As with personalities in general, individuals varied in their awareness of their own Interactional Identities. Only three participants seemed aware of their Interactional Identities to me when describing their work. Designers and developers generally applied stereotyping when each described the other. The implication for artefact use is that some meanings ascribed to artefacts may not be conscious and that designers may not be fully aware of how developers use artefacts and vice versa.

Designers and developers frequently interact in meetings. When creating software it does not seem to be important to “match” Interactional Identities; the actual pairings in

successful partnerships made little logical sense. This study hypothesized in fact that mismatches were best, as they produced contradictions that, when resolved, would result in a better product.

With regard to the stability of Interactional Identities, some evidence from this study suggests that Interactional Identities persist throughout a project and also from project to project. Interactional Identities also appear to be the same regardless of who the individual is interacting with. From the perspective of this research, the implication is that individuals subscribe certain meanings to artefacts consistently across interactions as individuals enact their Interactional Identity through their work.

With regard to changes in Interactional Identity, data from this study suggests that the Interactional Identity and the organizational culture are associated for designers, but not developers. This suggests that the Interactional Identity may change if a designer changes organisations. Interactional identities appear to evolve over the course of a career, but did not do so very rapidly. After approximately 10 years of experience both designers and developers seem to develop more expansive and less task-focused Interactional Identities. For developers this means moving from task to team issues and for designers the pattern was to move from an Interactional Identity that reflects frustration over their lack of control over the design work, to an Interactional Identity that is more efficacious and acknowledges and incorporates the role of other team members in the design of the user interface. Because Interactional Identity and artefact use are intertwined, the meaning of artefacts likely changes slowly over the course of a designer or developer's career from the view that artefacts help to accomplish a task to the view that artefacts support collaboration.

With regard to designer-developer interactions, the results show that both developers and designers develop a more nuanced view of who determines the user interface over the course of their careers. In general, after about 10 years of experience, designers and developers appear to come to see the user interface design emerging through teamwork. This is especially true for the designers who appear to have to “struggle with their ego” to begin to come to terms with the role that others play in designing the user interface because they initially consider that designing the user interface is solely their responsibility. This, of course, has implications for how designers and developers view artefacts. In general, in the first 10 years of a designer or developer’s career, artefacts are aids to performing subtasks of their primary design or development task. Later, artefacts are understood in terms of how they support communication and how they help to progress the joint work in a team context.

To summarize, the work of each designer and developer is coloured by his or her unique Interactional Identity. This is a reflection of how the designer or developer sees their joint work and sees the role of artefacts in their interactions. This analysis reveals that artefacts have another level of meaning to practitioners beyond their obvious function. A designer’s Interactional Identity is a factor in how the practitioner selects, uses, and understands artefacts that they use in interactions. In fact, the whole meaning of an interaction for an individual is coloured by their Interactional Identity. Interactional Identities were also shown to develop slowly over time for both the designers and the developers.

Results and Discussion Perspective 4: Mediated Interactions and Meetings

The work of designers and developers concerns the same object, so their work must necessarily be aligned. This part of the dissertation asks: How do artefacts support designer-developer interactions in meetings? To answer this question, I analyzed face-to-face meetings in which designers and developers participated. Such meetings were arranged by members of the software team to advance their collaborative work. I also analyzed the artefacts used in meetings. This analysis reveals how meetings advanced the work of creating software, what happens in meetings, and the role of artefacts.

Throughout these analyses, the emphasis is on the interactions of designers and developers in meetings and the artefacts that are used when designers and developers interact. Three studies are presented. The first identifies artefacts that support designer-developer interactions, drawn from data collected in eight organizations. The second analyzes 13 video-recorded meetings of designers and developers at work in four organizations. The third analyzes software development work in sequences of meetings in four organizations. Many images of artefacts are included in this chapter, and all have been altered to ensure anonymity of the relevant organizations.

Study 1: Shared Designer-Developer Artefacts

Artefacts were ubiquitous in all eight workplaces. This study focuses on artefacts that mediate designer-developer interactions. This study used photos of artefacts used in meetings (a subset of the 450 photos taken at workplaces and during artefact-mediated interviews) and screenshots from videos of meetings. Interview- and video-transcripts were also useful, as these contained many references to artefacts.

My entire corpus of artefacts from which this analysis is drawn comprised several thousands. In all cases, I examined artefacts in their context of use because all photos, images, and references to artefacts were embedded within meeting notes or meeting transcripts. The artefacts included a) many that were carefully constructed, such as drawings or diagrams, b) some that were ephemeral, such as stories, ideas, and questions, c) many informal ones, like to-do lists, and d) some that were merely references to artefacts outside of meetings, like software or abstract models.

The Development of a Categorization of Artefacts

Using coding techniques from grounded theory to categorize artefacts, several categorization possibilities were explored. One was to code artefacts by form e.g., lists, sketches, documents and so on, which was the approach I took when I began the coding. This approach was useful because it revealed some surprising artefacts (like meeting places that were purposefully designed for designer-development activity), but it also isolated the artefacts too much from their use. Another option was to code the artefact according to the sensory modality most stimulated in those who used the artefacts e.g., visual, aural, and experiential.

Visual and aural artefacts are self-explanatory. Experiential artefacts were artefacts that were experienced through multiple senses and over time. Although material in nature, such artefacts required the active engagement of an individual in order that they could be perceived in some way. Also, it was the experience of the artefact that was an important part of its perception.

Coding by sensory modality was very useful for revealing artefacts that were auditory, such as stories, and those that were experiential, such as interactive prototypes.

However, this sensory-modality categorization scheme had the undesirable characteristic of being too much about the subject experiencing the artefact. Another option I pursued briefly, but abandoned, was classifying artefacts according to the theoretical concepts of cultural-historical psychology. In that scheme I considered classifying artefacts that were transformative aids, that were the objects of transformation, and so on, but this would have been of questionable value because such categories are too general.

The categorization that was finally used was linked to the purposeful interactions that a group of artefacts supported. This seemed to be the right categorization scheme because it integrated together elements of the subjects, artefact, theory, and domain. This categorization scheme included artefacts that were both tools and the object of activity, and allowed artefacts to be coded in more than one category when they were used for multiple purposes. For instance, I saw an artefact at Telespace being used for planning purposes in one instance, and serving as a high-level model of the entire application in another context. This occasional need to code an artefact twice so it would appear in two categories is entirely consistent with the principles of artefact-mediated actions in cultural-historical psychology.

Below I list each artefact category in the final scheme, I present several typical examples in each category, and discuss the category briefly using quotes from interviews. The terms used for the categories do not necessarily correspond to those I heard in work places. To begin I provide definitions and a brief discussion of three meeting types considered to be important artefacts in this dissertation (because they are specially evolved tools that support design work), and then I describe artefacts used in these meetings.

Meetings as Artefacts

Cole (1997) defines an artefact as follows:

“... an *artifact* is an aspect of the material world that has been modified over the history of its incorporation in goal directed human action. By virtue of the changes wrought in the process of their creation and use, artifacts are simultaneously ideal (conceptual) and material. ... the properties of artifacts apply with equal force whether one is considering language/speech or the more usually noted forms of artifacts such as tables and knives which constitute material culture. (Cole, 1997, p. 117)

In this passage, Cole suggests that culturally-created artefacts can be found in their “more usually noted forms,” i.e., their visual and tactile forms, as well as in speech. I suggest that experiential artefacts, such as structured forms of interacting, (i.e., meetings) can also be artefacts. Because meetings are cultural legacies that have been modified over the course of their use in all kinds of activities, including software-creation work, they are cultural tools. Further, because meetings that advance software creation work have evolved for this purpose, they are a tool or artefact in software creation activity.

In this dissertation, meetings are seen as cultural artefact that are used to achieve a purpose, although their purpose is not achieved in the course of a single individual goal-directed human action as Cole suggests in his definition of artefacts, but rather across the numerous goal-directed interactions of the multiple individuals engaged in a meeting. Meetings, complex and dynamic artefacts, are also seen as “container” artefacts for other artefacts that are brought to and developed in meetings.

Designers and developers frequently suggested holding meetings to advance their work, and meetings were ubiquitous across workplaces. I observed designers and developers interacting in scheduled meetings in prearranged locations. These meetings

involved other team members and sometimes the entire team. Because I also observed deskwork, I noticed that it was frequently punctuated by chats with colleagues and occasionally by impromptu meetings. Impromptu meetings happened when two or more colleagues quickly arranged to “work through” an issue in a readily available meeting place. Work-focused, desk-side chats typically lasted 15 minutes or more, although one was half an hour long. I considered chats to also be a type of meeting because people were interacting, the topic was work, and the matters that were discussed and the artefacts that were used were much like those used in scheduled meetings. However, compared to scheduled or impromptu meetings, work-related chats were less prolonged, they were more common, and happened more frequently.

The number of meetings varied across the four organizations where I observed; the variation is presented in Table 28, which shows that 117 meetings were observed in total and of these, 46% involved at least one designer and one developer. In these calculations, project managers were not counted as designers and developers, even if they were experienced in those roles, as this was not their primary role on the team.

These data show that that designers and developers interacted very frequently and through various meeting types, but that designers and developers also interacted with other specialists on their teams, because about half the meetings I observed had just one designer or developer in attendance. Given the size of the teams (a mean of seven people), these data indicate that for the types of projects I observed, designer-developer interactions were a common occurrence.

A careful count of all the meetings revealed differing proportions of scheduled meetings, impromptu meetings, and chats at the various sites. Talia stands out in its

exclusive use of scheduled meetings, contrasting with Telespace where there was a strong preference for communicating through chats. Jaba team members engaged in many chats, but they also had some scheduled meetings and more impromptu meetings than any other organization. Quadrow used all three types of meetings in roughly equal proportions. The pattern of meetings appeared to be influenced by specific product challenges (e.g., game design appeared to require a lot of face-to-face interaction), the organizational culture (e.g., workplace rules about chatting) and aspects relating to the physical workspaces (e.g., the availability of meeting rooms).

Table 28 *Meetings at Talia, Jaba, Telespace, Quadrow*

Meeting Type	Talia	Jaba	Telespace	Quadrow	Total
Scheduled meetings	8	5	6	7	26
Avg. attendees	6	3	8	5	5.5
Both attend (#)	7	3	2	6	18
Both attend (%)	88%	60%	33%	86%	69%
Impromptu meetings:	0	11	5	5	20
Avg. attendees	0	4	3	3	2.5
Both attend (#)	0	4	1	1	5
Both attend (%)	0%	36%	20%	20%	25%
Work-related Chats:	0	28	32	11	71
Designer-developer (#)	0	13	13	5	31
Designer-developer (%)	0%	46%	41%	46%	44%

Scheduled meetings. Scheduled meetings were a structured way of organizing communication. Scheduled meetings had a prearranged meeting place, a prearranged list of attendees, and a declared purpose. People playing various roles attended. If there was an agenda, the meeting was most likely to be a regularly scheduled project meeting in which the team systematically checked the status of the project work. Meetings

concerning software creation, i.e., the majority of the meetings, whether scheduled or not, did not have agendas.

Scheduled meetings were used for a variety of purposes. For example, at Jaba, the lead game designer used scheduled meetings for communicating vision and direction. Talia arranged large meetings with many stakeholders to discuss a specific aspect of the project. Quadrow organized daily stand-up meetings, as did Talia, although I did not observe Talia's stand-up meetings. Quadrow also held scheduled meetings with clients. Telespace used scheduled meetings for managing the project by conducting project-wide status reports with the entire team.

Impromptu meetings. Impromptu meetings were opportunistic gatherings of designers and developers in a purposefully designed meeting space that spanned half an hour or more. These meetings were purposeful, geared toward the work of creating software, but never had an agenda. An alternative to a face-to-face impromptu meeting was an email chain, which I witnessed only occasionally, as all of my participants preferred to work face-to-face.

Jaba held mostly impromptu meetings, or what Owen called "working sessions". At impromptu meetings people offered their opinions on the software product under development, and the people who attended wanted to be there. These meetings were about "touching base" (Owen), making misunderstandings visible, "airing" ideas and problems, setting and resetting the direction for the project, and re-motivating the team. Prototypes were sometimes developed in impromptu meetings, but more usually a joint vision of the game was developed.

At Quadrow, impromptu meetings were for explaining the design, and “talking around it.” Mason, a designer, used impromptu meetings to explain “what the team should do” (Mason). He also used impromptu meetings to test the fit between scenarios and mock-ups, which he says helped clarify the project scope. He held meetings around “tickets” to check if he’d missed anything. Muriel, also a designer, said that holding an impromptu meeting supported discussions about how something was going to be implemented, and that they were also valuable for getting input from the developers that might result in changes to the design. Wil, the developer at Quadrow, said that impromptu meetings were usually design sessions and mock-up review sessions.

Chats. Chats were informal, relatively brief discussions. They were a relatively unstructured way of meeting that was usually a spur of the moment, often desk-to-desk interaction between two or three people. Occasionally I saw chats in the form of email exchanges.

I saw many designer-developer chats in the four organizations, except in Talia. Here, the designer and developer, who worked on separate floors, were much more likely to exchange emails.

Artefacts Used in Meetings of Designers and Developers

This section presents a categorization of artefacts used in the meetings of designers and developers, which were also regularly attended by other members of the software-creation team. Because of the diversity in the attendees of such meetings, the artefacts used were accessible (i.e, meaningful, easy to use, and to understand) to individuals from a multiplicity of disciplines. Further, because the stage of the software work was concept design work, all of the artefacts in these meetings generally assisted in

determining what the team would build. While many meetings were observed, I was also told about meetings and meeting artefacts in interviews with designers, developers, and managers at four organizations where I did not observe. These data were also included in this analysis.

In general there were no shared specialized software tools used by the teams during the meetings I observed. Outside the meetings the teams' shared tools were code repositories, email, and meeting-room booking software. One notable exception was Quadrow's ticketing software, which all of the team members at Quadrow used to coordinate their work. Many of the artefacts described here were the output of discipline-specific tools, but the specific tools used to generate the artefacts were of no consequence to the meeting attendees and are therefore not discussed here. The classifications are summarised in Table 29.

Table 29 *Brief Description of Artefacts Used in Meetings*

Artefact Category	Description
Software under development	This artefact helps team members to reflect on the end-user experience. It may be in a limited form and contain placeholder elements; this includes prototypes, mock-ups, and intermediate versions of the product.
Interface Proxy	An artefact that stands in for some aspect of the interactive product, e.g., look and feel, navigation, flow of interaction, and so on.
Context Proxy	An artefact that explains who, when, where, why, and how the software will be used.
Computational Proxy	A stand-in for the computational back end of an application.
Specification	A comprehensive description, e.g., of requirements or a design of the user interface.
Plan	A statement about intended future work.
Exemplar	A concrete artefact that is like the application being designed in some way.
Framework	A software structure or digital artefact that supports the development of other software.

Model	A mental model that guides the software-creation work.
Idea	Suggestions for evolving the interface or application.
Question	A potentially challenging inquiry that prompts reflection.
Story	A narrative construct that binds together a character, action, and sometimes a setting. Typically used to explore how an application will be used, but narratives have many purposes.

Software under development. The software under development is the object of the software-creation activity. Traditionally, software products are generated from source code, but especially in cases where software has a significant user interface, the software under development is generated not only from code, but also from input files such as XML files or other types of content files that specify text, images, video, or audio. The software under development consisted of all of these elements bundled together and then presented as a whole in meetings. The software under development then, if used in a meeting, would give a good idea of how the software would be experienced by an end user if released on that day. An example of an artefact that was experienced as the

Figure 13. Example of software under development. Jaba's game prototypes were so realistic it was treated as if it was the software under development. This particular prototype had placeholder elements for graphics, art, and content, but was still a playable game. Gamers could interact with the elements, select characters and perform actions.

software under development is shown in Figure 13.

Four of the eight organizations studied were developing software “from scratch” (Jaba, Quadrow, Onu, Kadel), three were redeveloping software, or at least user interfaces, from scratch (Talia, Telespace, Zambee), and the last organization was developing a new feature in an established product (Edgeworks). Once the developers had begun coding the software, many references were made to the software under development. Typically the software was mentioned to report on it, or to check its status.

However, two organizations that were developing software “from scratch” made efforts to produce very realistic prototypes of their product so it was “as if” the software under development was in the meeting, even though the final version of their product would be built using a different technology.

At Jaba, the developer was constantly recreating a prototype of the game under development. He used a prototyping tool and worked to ensure that the prototype was as close as possible to the currently envisioned game. The prototypes he created were essential to the work of the team. In fact, the team only became motivated to develop PlayLearn-it after the lead game designer had designed and the developer had implemented the first prototype of the game. Thereafter, the designers who produced various aspect of the game (i.e., the art, the graphics, and the learning content) had their contributions rapidly incorporated into the prototype by the developer. The combined prototype was then re-released to the team, almost on a daily basis.

Quadrow also developed a realistic prototype for their Web application, but with a different tool. In Quadrow’s case, the designer, and not the developer, was constantly

reproducing it. In both Jaba and Quadrow's cases, the flow of interaction and the integration of many elements with each other were important considerations in the work.

As valuable as such artefacts that represented the software under development could be, producing an artefact to represent the software object was no easy task. At Kadel, the developer reported that during concept design work he continually reproduced the software, but from his perspective, this task was daunting because it was difficult to keep up with the designer's and business person's ideas.

Interviewer: What are the other big issues?

Jack: Ah, there's one for me [Laughing] that's a very big one. The, ah, as many ideas as people can have, I'm the only one who's able to actually build them. And ah, so that, ah, is a real bottleneck.

In organizations where the project work was well underway when I arrived, and where I focused on only one part of the development work, I noticed that there were reasons for the current implementation of the product (i.e., the software under development) to not appear directly in meetings. For example, the Telespace team I observed working on a reporting feature had developed software for their product, but the designer did not even look at it because it was too volatile.

Jay: the next phase of my activity is now to go back and actually get a working copy and actually see what's happening with the designs. ... We have not had time to even look at [version] 9 [of the software] even, and 9 was really rocky. Um, [version] 10 is settling down.

Interviewer: So it might actually make some sense to look at 10?

Jay: It might be better to wait. ... Wait till 10's finished, which will be the end of this year. So we're talking, starting January it might be a good time to look at [version] 10 [of the software].

So it seems, that while the designers and developers made many references to the software under development, it was unusual for the software under development to

appear directly in meetings, except in the form of a very realistic prototype that was treated as though it was the software.

Interface Proxies. The interface proxy was by far the most commonly occurring artefact. An interface proxy stands in for the object of the activity (the interactive software). In one meeting one would normally find multiple interface proxies, each one emphasizing a different aspect of the interface (e.g., visual elements, the dynamic interaction, and so on) or each one representing an alternative design solution.

An example of an interface proxy was the game shown in Figure 13. Other examples are shown in Figure 14 through Figure 18.

Figure 14. An interface proxy. An interactive prototype of a gaming element being designed at Jaba.

Figure 15. An interface proxy. A prototype developed by a contractor to show the visual aspects of an interface at Quadrow. This design constrained the look of the Calcul-it software.

Figure 16. An interface proxy. A whiteboard sketch of two connecting screens drawn by the designer at Quadrow to show some new behaviour in the user interface.

Figure 17. An interface proxy. A prototype developed by the designer at Telespace using Powerpoint. The prototype shows the screens to be developed and the flow of interaction between the screens. This prototype was mostly used in its paper form, and was carried from desk to desk for chats between any of the members of the software team.

Figure 18. An interface proxy. A drawing created by Talia's designer. Typically the designer would create two or three prototypes for each meeting. Each prototype presented a different concept or idea.

When asked about product proxies, Mason, a designer said,

Well, basically um, it's a focal point for people to discuss, right? So you're basically creating something, it might not be the perfect thing, or you might think it is the perfect thing, and then everybody looks at it and goes like: "Oh my God, that is what I'm talking about," or, "I'm not talking about that at all," or wherever, right? So it's a way that we can propose new things, and for everybody to look at it, and it's a concrete thing to look at and they can...they can actually make comments on it or about it. (Mason)

Damian, the developer on the same team as Mason, saw product proxies as useful because they incorporated Mason's concepts or ideas.

Interviewer: So [you say] his [the designer's] concept diagrams are the most useful. So why do you think that is?

Damian: it's...the interaction ...it's more sequence-like, right? He would present, you get the concept, you get to say. Once he's done then it's your turn to get more information, or [to] question some of the concepts. Right? So therefore, like, I don't think any other document would be useful ...when I'm interacting with them [the designers] ...

Context Proxies. A context proxy is a design artefact that supports the software team to access the product context environment directly or indirectly. Indirect access to the context was achieved through the use of documentation or visualizations, but direct access to part of the actual product context also occurred. For example, direct access happened in the case of two teams who were designing for Web users and who could easily bring the context for their application into the meeting room, simply by browsing to sites where their applications would be incorporated.

One designer, Kate, from Kadel, discussed the need for what she called "upfront analysis" and the lack of documentation about her end user's tasks and the context in which her end users would need the tool that she was designing. She explained how this

caused problems when she prioritized her work. She talked about her expectations that such a context could be captured in a document.

Kate: So, one of the things that has changed ... is that we do a lot of upfront analysis. Because at the time, I had no documentation to go on about, um, task prioritization. Like, so the user's gonna come here-everything, according to the FEs [financial experts] was important. Everything is important. So if you-if I was to take an email analogy: setting up the email servers is just as important as setting up-as reading an email. How often in your email program do you go and set up your service? Probably none.

Another designer, Nick, also from Kadel, talked about how he learned about the context in which the software he was designing would be used by sitting in a meeting and having the feature analyst (the business representative in his organization) explain and sketch the context of the application to the team.

Nick: So a lot of it is, um: sitting with ah, the owner of the feature, usually one of the financial engineers in our cases, or the business analyst-sitting in a room and doing a lot of 'whiteboarding'. Um, we do quite a bit of 'whiteboarding' about, you know what it is that-what's the entrance and what's the exit? Like: why am I [the user] here and what is it that I [the user] need to do and what it is that I [the user] need to get out of this [interaction with this software].

Telespace had an entire lab set up to bring the context to the designers and developers. In this heavily used lab, the previous version of the software was running in a quasi-realistic setting, and a number of chats between the designer, developer, and designoper ended up in this lab in order to incorporate aspects of the context into the discussion. Talia had the previous version of their software running on a server, and had it populated with "toy data" so the designers and developers could "play with it."

Computational Proxies. The computational proxy was an unusual artefact seen only in two workplaces. This artefact stood in for the computation engine to be designed as part of the back end of the software application. It supported a discussion about the

behaviour of the application, and was only required when this behaviour was complex and would impact the user interface design. One example of a computational proxy is

Figure 20. A computational proxy. This drawing was created at Jaba, with paper, pencil and ruler. The team played this game with a penny and dice. The purpose was to explore the variables in the game and to determine how the variables should be used together.

 A hand-drawn table on a whiteboard. The table has four columns: MIN, QUANT, \$, and ORDER. The rows are labeled A, B, and C. The data is as follows:

MIN	QUANT	\$	ORDER
A	200	\$0.1	1
B	400	\$0.05	2
C	50,000	\$0.01	3

 Below the table, there is a handwritten "END" and a small box containing the number "10".

Figure 19. A computational proxy. This table drawn on a whiteboard was created at Quadrow to simulate a computation. The computational engine at Quadrow was very complex. Work with this artefact addressed the scope of the computation, and therefore the scope of the application. Quadrow also had a spreadsheet programmed to simulate their computational engine.

shown in Figure 20 and another in Figure 19.

Specifications. A specification is a comprehensive description of something such as the requirements for software or a detailed design description of the user interface.

This is developed as a shareable artefact that generally has a set framework. I have included requirements documents, design documents, and content documents (i.e. content

templates) into this category. However, sometimes the specification is not a document itself, as was the case with the document templates I was shown in one organization. A specification must give both broad and detailed direction for the work of the team or for part of the team. A specification would not be mistaken for the product i.e., a product proxy is not a specification, and designers and developers do not use specifications as though they were the product.

Plans. A plan is a statement about intended future work. Plans sometimes capture important timeframes for the completion of the work, and sometimes people are assigned to tasks. The occasional plan was very complex as shown in Figure 21, but most plans were very simple, and were in the form of a to-do list. Some plans had unusual forms as in Figure 22.

Although plans did appear from time to time, intricate plans did not figure strongly in meetings for creating the software.

Figure 21. A project plan. A traditional project plan used at Telespace. Tasks for developers, designers and designopers are in rows, time (days) are in columns. Data in the table included planned start and completion dates, current status, resource use, and current activities.

Figure 22. A project plan. This drawing served as plan at Telespace. Each little box represented a feature; the designers tracked how many of these were completed using this diagram.

Exemplars. An exemplar is a concrete artefact that is like the application being designed in some way. It generally represents a positive example of some aspect to be emulated, although the occasional negative exemplars appeared too. Negative examples were often the previous versions of products, as was the case at Talia where the team explored the previous product in order to create a list of elements that should change. Exemplars were typically other software products, often the organization's own product. As Jason says, "You're just leveraging what you've done quite well to create a new game." (Jason).

Sean, the Talia designer, describes how each person on the team is influenced by different exemplars, and how in his experience developers used them to influence his design to be more or less novel.

When we get into like what the ... form's gonna look like or what the tree's gonna look like, we're gonna be proposing stuff that we [the designers] like [which is generally more simple designs]. We have those axes to grind, you know, like the bag of hammers that we have that we carry around with us and then we...we're gonna throw some hammers on the table and say, "Well, do it this way." And they're [the developers are] gonna go like, "Well that's not the way YYY looks" and-like, you'll get the reverse argument [the argument for a more novel design] then, right? (Sean)

There were generally very few exemplars for each of the products, but they appeared to be influential. However, occasionally exemplars were used a lot. For instance, seven different exemplars were used in one meeting at Jaba. These included references to a previous Jaba game where the gaming world was reduced to simplify the work of the artists; how logs worked in Baldur's Gate and Neverwinter Nights; two previous Jaba games and their rewarding aspects; a previous Jaba game and what it does when the player fails; and Microsoft Outlook and how it can appear big or small.

Frameworks. A framework is a software structure (e.g., code) or digital artefact (e.g., XML document or a digital image) that supports, and is the starting point for, the development of other software. As a digital artefact, it is amorphous. Frameworks ensure certain aims are met in the completed software.

Framework examples were extremely varied and because of their intangible nature they were generally just referred to in meetings, i.e., appeared in the talk. Because frameworks were frequently part of the fabric of the work, and assimilated into the culture of the workplace, they were also only mentioned rarely.

Frameworks were observed in all workplaces. Damian's framework code for the user interface at Talia was an example of a framework under development that would ensure the entire interface employed the designer's work, and that the software was

maintainable. The software interface to the server back end in the Talia project also acted as a framework for Talia's designer because this was the designer's source for a clear specification of interface objects. Designers Sean and Muriel at Zambee used CSS style sheets to ensure accessibility standards were met in their Web software because this approach meant the standards became a part of that software. The CSS style sheets therefore also acted as a framework. Jack, the developer at Kadel, studied and carefully selected the right application programming interfaces (APIs) for the development work on the user interface. Jack also worked within the constraints of the Facebook API, which he and the team knew very well. Frank, the designer at Edgeworks, considered that each software iteration in their Agile-like process provided a stable foundation for future development, and so the software produced in an iteration was a framework for him. Frank at Edgeworks had a repository of software elements from which he could make user interface components and he used these to build standard layouts for screens, which he shared with the developers, so these were an example of a designer-created framework. Eric's templates for different HTML page types that were possible contenders for course development work in Zambee also satisfied the criteria for a framework.

In all cases the digital artefacts mentioned provided both a foundation for, and constraints on, the work of the team, both of which were valuable in advancing the work.

Models. A model is an abstract artefact that guides the development of the application. These could capture a rationale, general knowledge, or a process, and are the type of entity that might appear in a book or a course. Models often influenced the overall product and represented a direction or a position to take.

Examples included a domain model, such as a specific gaming model, or specific learning model, or a model for the type of software being developed (e.g., a content management system). Models were usually articulated in words, but sometimes appeared as diagrams or sketches. They often come from outside sources, such as academia, and were powerful influencers on the software's direction. Figure 23 and Figure 24 are two models mentioned in meetings, which I did see sketched out in the workplace—the first for a paper the organization was presenting at a conference, and the second was used in an explanation to a person new to the workplace.

Figure 23. A model. A model of Bloom's taxonomy that supported the Jaba content/learning designer's thinking about learning in the game.

Figure 24. A model. A model of flow that supported the Jaba content designer's thinking about the gamer's experience. In the designer's opinion, gamers should be "aroused, in a state of flow, or in control." Any other state would be undesirable.

Models could be generated within the team as well. As Wil says,

And, once that's done [the mock-ups are created], um, myself, the developer and database analyst would...would start looking at the design overall, ah, high-level architecture, what the application would look like and the database schema on the database side.

Another example of a model occurred when Jay talked about the design lead who created "structural diagrams" to guide the work of the entire team:

Jay: He's the customer advocate. He's the um, he's the retail expert who understands not only the customers' needs but also the ... hardware and how software should be arranged . .

Interviewer: But he also does some design of the system. But not at the level that you work at, right?

Jay: It depends. Sometimes he'll dabble. Sometimes he'll say: "That button should be there." Mostly he's structural.

Interviewer: Big picture?

Jay: Yeah. Screen flows, task flows.

Ideas. Ideas were simply concrete suggestions for evolving the interface or application. There were, in general, three categories of ideas: positive ideas, which were

suggestions for shaping or enhancing the software interface; negative ideas which were suggestions for eliminating something from the software interface; and constraining ideas which were a suggestion that would constrain the future work on the software interface in some way. These latter ideas were like rules. I did not include ideas for how to complete software work in this category, as these were far less common in concept design meetings.

Many examples of ideas were found in the talk of the designers and developers. As Jason says, “I think talking to someone will always be a better way of communicating obscure and abstract ideas,” but sometimes ideas were expressed in sketches. As Jason says, “If it’s something that has to be drawn, then it has to be drawn, right? Like if it...I think if it’s gonna take you more than 3 sentences to describe ...”

Wil described how they used sketching at Quadrow.

Wil: We [the developers] use that a lot; I do. Ah, I know [the designer] does, too. In some of our meetings, he’ll get up and draw something.

Interviewer: So what’s in those drawings? When do you feel you have to go for the marker and start sketching? What brings that on?

Wil: I don’t know, sometimes, a picture is too...far easier for people to understand, makes it clearer. It’s always at the high level-right?-relationship between objects, or relationships between pages, like navigation, or...or what’s gonna be displayed on the page, things like that. ... Sometimes you wanna just ...brainstorm through it to see if it makes sense. And then, once you’ve ... decided on a particular path then, yeah, (it’s all)...everybody lines up on the same page

An example of a positive idea is: “... we need a new type called ‘don’t know’, the ‘are you feeling lucky’ type” (Mason). An example of a negative idea is: “We’re going to have to remove this facilitator idea. As much depth as it would give us, we’ve got to get rid of it.” (Owen); An example of an idea that is a constraint is “If they [our clients] are doing it the wrong way, we could always flop the engine if we have to, to

model it, you know, we could always change their rules around, but we should model it the way the consumer would expect it to be.” (Mason).

Questions. Many questions were posed about the design situation. These included questions that prompted reflection, challenged an assumption, challenged a gap, or challenged a capability. Not all queries were coded as questions. Not coded were questions that occurred in all kinds of talk, such as questions that confirmed an action; questions that confirmed the state of a task; questions that confirmed an assumption; questions about what do to do in the next moment; questions that confirm what was said, seen, or exists; or questions that are a request for information. Some questions that were more domain-specific were also eliminated: questions about the wider process, questions that were off-topic, that were an idea posed as a question, or questions that were a rule posed as a question. Questions test ideas or models. Questions find the limits of ideas or models and clarify them.

Questions were usually spoken, but occasionally a question was posed with an artefact, as in the case at Kadel where a designer asked about problems to do with screen real estate by using two artefacts, one overlaid over the other. Another such example was at Edgeworks where the developer created an artefact showing that the feature, as designed, would not support multi-lingualism.

Examples of questions include: “Isn’t that part of the game? Isn’t it that you should be able to keep track of everything?”; “The other three areas, do they fit well?” “What are you going to do with products that don’t fit really well?” “You’re teaching them to tie actions to employees. Is that right?”

Stories. A story is a narrative that has a main character (not necessarily human), an action, and sometimes a setting. It is concrete in at least one of these aspects. Stories frequently began with words like “So, let’s say, say, you ...”, “they ...”, “I ...”, “if,” “like,” “so if,” “maybe ...”, “then when,” “so let’s imagine ...,” “so now ...,” or “as a player I might ...”. Stories often end in “Right?,” or “Ok?,” a request for a confirmation. They vary a lot in their richness, but are generally fairly short. Here’s a typical story: “So let’s say that I’m at the very beginning [of the game] and I’m going to [select the] create [button]. Ok? So I pick one of the things for the actors [in the game] to create.” (Owen)

Although usually the main character in a story is human, this is not necessarily the case. The game or the application or the calculating engine could be the actor in the story. Examples from Talia include “if our results return within a page, we can sort within that”; “a page could be a hundred items”; “they’re [complex filters are] so hard to use right now”; “so, if your [software] manager managed more than one object type if there happened to be one then ...”.

At Quadrow I encountered stories in tabular form that were supplied by clients as test input for the calculating application the team was designing. The tables supplied about 20 cases (one case per row), each case representing a different scenario for the calculator. Precise values for half a dozen variables (one variable per column) were entered in the columns of the tables. The rows represented real-life cases the clients expected the software to be able to compute. Cases were used to identify important narratives to consider, to identify boundary cases, to exclude cases that were out of scope, and to address cases that needed special handling.

Mason explains that stories helped to cope with the complexity of the project work at Quadrow and helped the project team to understand the problem domain.

Interviewer: What are your avenues for dealing with complexity?

Mason: I think, you know, like, we came out with that meeting with, I said, we need to create, you know what I mean, a document where we have these [a whole bunch of scenarios] in it, right? And we can run those against the mockups, right? Which was what we were kinda coming out with because it was, there was, too many permutations, so... Creating a ... representative sample that we could use ... it was getting out of hand, ... so that's why I said, "We need to define these things." It's kinda like analysis, right? We need to pick out-we need to create the problem domain before we can solve it, right? And then run that against our screens to see if you can fill them in right? Or are we missing something? It's the only way ... still-it's probably enough ... There's diminishing returns [with more scenarios] after a point, right? We do need to get enough of our representative samples but not go too crazy ...

Although Mason expressed the most important reason for using stories, which was to articulate the context of use in a sufficiently detailed fashion that the project team would understand how the product would be used, there were also many other reasons to use stories, as will be noted in Study 3.

Summary

This study presented a categorization scheme of artefacts found in meetings where designers and developers interacted. The scheme was domain-specific and presented a number of artefact types that assisted the software-creation teams in determining what to build. Although the specific artefacts and types of artefacts differed from one organization to another, all organizations were remarkably consistent in that the designers and developers used the artefacts in almost every category described here in their concept design meetings.

Study 2: Development of Creative Work from Meeting to Meeting

I observed a significant amount of solitary work by designers and developers at their desks; at these times work progressed steadily. However, at meetings significant adjustment and development of the object and objectives of the activity occurred.

Table 28 showed that different software creation teams regularly engaged in three different types of meetings: scheduled, impromptu, and chats. Using meetings as a unit of analysis, and interaction analysis as an analytic technique, this section reports on the designers and developers engaged in software creation work at Jaba. This study addresses the development of this work from one meeting to the next, explores the general structure of meetings themselves, looks at the nature of transformations that occurred in meetings, and examines the role of artefacts.

The data used for this analysis were photo-annotated notes of meetings, and video recordings of meetings. Because this was fieldwork, many of these meetings did include other team members, and this analysis accounts for this reality. I looked for key attendees, nonkey attendees, the major tension addressed in the meeting, and how the key attendees supported their position through talk and artefact use. For each meeting I identified artefacts that aided in the transformation, artefacts that were transformed in the meeting (if any), and the outcome of the meeting.

The outcome of an analysis of a series of meetings was a large Excel spreadsheet comprising eight pages of data, not shown here because of its size. Important elements of each meeting were captured in this spreadsheet, which allowed for a view of all Jaba meetings at once. For each meeting, I examined beginnings and ends, the major processes

in the meeting, and also the artefacts used in the meetings. The goal was to identify the major transformations within a meeting.

I present a full case study of the work of designers and developers at Jaba. Seventeen meetings at Jaba that addressed the development of the PlayLearn-it game were analyzed; these meetings all addressed the development of the product I was tracking. Of these meetings, only meetings that had at least one designer and one developer are reported here. Further, to focus on the most important meetings at this organization, I do not report on chats. Six meetings are therefore reported of the 17 I analyzed. In describing these results, I refer to artefacts using the terminology of the team, but associate these with the classification scheme developed in Study 1 by indicating the type of an artefact in parentheses where this is not entirely obvious.

An Analysis of Jaba Meetings

Jaba creates games. The software creation team for PlayLearn-it consisted of four types of designers, an artist, a writer, and a developer. Part of the work of creating PlayLearn-it involved developing a virtual world that was meant to be a realistic setting. There would be a number of characters in the game, and the gamer would play the game by controlling how the game characters interacted with each other. This game was a learning tool; the gamer was meant to be learning how a business process in the simulated setting could be conducted efficiently, while the gamer had fun playing the game. Therefore the *content of the game*, meaning what the characters said, how they acted, and the consequences of their actions, had to be realistic. I followed the lead game designer, the content/learning designer, and the lead developer or programmer in this organization.

At the beginning of the series of meetings reported here, the requirements document and the design document (two types of specification documents) for PlayLearn-it had been written. However, the requirements document, written by an analyst, and the design documents, written by the lead game designer, had been ignored by the artist, graphic designer, and developer. The only person who had read the documents was the content/learning designer. The content/learning designer, in preparation for her role as the person responsible for “adding content to the game” had extracted a large amount of data from the subject matter expert (SME) in the form of an unformatted, textual, content document. The very first game prototype, called the Sketchbook prototype by the game developer, had just been released (this was the first version of the software under development). This had been designed by the game designer, and implemented by the developer. Team members were excited because they knew the initial prototype would be very basic, and they looked forward to contributing to the design of the game.

The first version of the game implemented the ludic elements of PlayLearn-it, in this case, an unrecognizable variation of the rock-paper-scissors game. The game was very basic. There were placeholder characters in the game and some functionality; the gamer could assign actions to characters in one of four envisioned game stages, as shown in Figure 25. The number of characters, stages, and actions had a large impact on the amount of content that would be required for the game, an issue that was very important when it came to completing the game on time. That issue was addressed at length between the learning and content designer, but only peripherally here, as the number of characters and actions and the amount of content did not affect the programmer much,

because he had no difficulty adding and deleting characters, actions, and content. The other members of the design team had all played the game for the first time; and, although no one was favourably impressed, no one had said anything to the game designer yet.

The series of meetings reported here addresses the development of the basic framework for the game, which was important because this game was going to be the first in a series of games of this type. Meetings also addressed the development of a new gaming element called an “action selector component” that would be the basic mechanism through which gamers would control the action of the game. Both of these issues had a large impact on the work of the designers and the developer. Table 30 provides a summary of the meetings.

Meeting 1, a game with no narrative potential. In meeting 1, the writer, who would be responsible for the game’s narrative flow, arrived with a written list of questions about the game’s potential to support a game narrative. She said she foresaw problems developing a game narrative for the current game prototype (i.e., the software under development). The lead game designer, with whom she had arranged to meet, used the prototype shown in Figure 25, which was run on a laptop in the meeting, to explain his rationale for the design of the ludic elements of the game, which were his responsibility. He also explained a number of game rules that were not evident in the prototype. The primary tension that triggered the meeting was between the ability of the game to be extended with a narrative, and the “gameiness” of the game (the team wanted the game to feel like a game that players aimed to win and not a simulation that players merely experienced and they used the word “gameiness” to capture this intention).

Figure 25. The sketchbook prototype used in Meeting 1.

Other attendees in this meeting were the content designer, the developer, and a designer who had just been hired the day before. All three were eager to hear the writer's questions and the game designer's answers and had asked to attend the meeting for this reason. The meeting lasted about an hour, and artefacts were used to transform the team's understanding. The writer's list of questions gave the meeting its structure (i.e., they functioned as a plan for the meeting) and purpose (i.e., the questions evoked reflection in the team). The writer's questions related to characters and their personalities, game chronology, the role of the gamer in the game, gaming actions, how to win the game, game strategies, and the function of random events. A prototype of the game functioned as the software under development. A Microsoft Excel spreadsheet of game character descriptions (i.e., an interface proxy) described the game character's personalities, which was suggested as an aid for the writer. A whiteboard sketch (i.e., an interface proxy) depicted what feedback scores would look like in the game.

The meeting did progress from the presentation of an initial problem, but not all the way to its resolution. Further, many other issues were also addressed. And, not everything that was discussed or proposed was helpful. In particular, there were a number

of artefacts that didn't seem to help in this meeting. The writer rejected a table of allowable game actions (i.e., an interface proxy) presented to her because she said she needed to understand game actions "in words." She also rejected an XML-structured content document (i.e., a framework) that contained details of random events that would be programmed into the game (i.e., an interface proxy). She explained that if she was going to be able to process information she needed material packaged, not separately in spreadsheets and text documents, but "everything at once" (i.e., she wanted a version of the software under development). One designer suggested the team use Myers-Briggs personality types (i.e., a model) to create the game character's personalities, but the team did not "pick up" this idea. However, while it is important to note that artefacts used in meetings may not actually be useful, henceforth, I report only on artefacts that appeared useful and supported productivity.

The meeting's outcome was that the Sketchbook prototype, as currently designed, would be difficult for the writer to work with because it lacked narrative potential. The other attendees, who also participated in the meeting by providing further questions and information, deepened their understanding of the game and the work ahead. For example, for this initial prototype, minimal placeholder content had been programmed into the game by the developer and then reverse-engineered into an XML file so that a text file of the placeholder content existed. Through inspection of this file, it became apparent that it did not align with a content file being produced by the content designer. She had assumed an erroneous model for the game play because she had not been involved with the development of the first prototype. Consequently, she had unfortunately collected much more content than was necessary for the game from the team's SME, an employee of

their client. Also, it was discovered that the content was formatted using erroneous assumptions. Therefore, because the designer and developer's work was not aligned, it became clear that the current content and game mechanics were incompatible. Also, seeing the Sketchbook prototype (i.e., the software under development) made the lack of learning mechanisms, art, and graphics readily apparent. More importantly, after the meeting, it was clear to the lead game designer that the team did not understand the game and were not comfortable with it.

Meeting 4, a new game model and an action selector component. Over the next weekend the lead game designer addressed some of the issues raised in the previous meeting by “inverting” the game. In the inverted game the gamer would cause characters to act by clicking on them and then selecting an action from a popup menu, rather than selecting the most appropriate character for a proposed action. In meeting 4, the lead game designer wanted to explain his new ideas for the basic game model to the team. This game, he believed, would support a game narrative better and would be produced in a timely manner because he had also made some simplifications, such as reducing the number of characters. The following artefacts, drawn on the whiteboard during the meeting, helped the game designer describe the new game: (a) a list of the 1st prototype's flaws (i.e., a list of bad ideas), (b) sketches of the new user interface (i.e., the interface proxy), (c) lists of rules (i.e., good ideas) for the new game prototype (i.e., the software under development), (d) a model of actors and actions that captured the basic game concept, (i.e., a model), and (e) a plan that showed that the game's model mapped to the three other game stages, shown in Figure 26.

Figure 26. A plan for how the game's model will map to the three other game stages.

The attendees in the meeting were the programmer and the content designer who were expected to provide “push-back on the design,” which they primarily did by asking questions. The contradiction addressed in this meeting was the design of the new game model versus the design of other game elements (the ludic elements, the learning aspects, and the game's fun factor). The game had been changed in response to the writer's concern that she would not be able to write a good narrative for the game. Now it was time for the team to ask whether the evolved game was still a game (this question, although strange, arose because they could not understand the game play), whether gamers would learn by playing the game, and whether it would be fun.

The meeting lasted two hours. Together, the designers and developer evolved their ideas about the game. For instance, they decided it would be built upon two patterns of the ‘rock-paper-scissors’ game (i.e. an exemplar), and they proposed a gaming motif “It's a matching game!” (i.e., a model). Also, they began to design other gaming elements besides the ludic ones, such as the action selector component and the way scores were to be displayed. This occurred through the development of a set of loosely associated artefacts that were produced throughout the course of the meeting including (a) a sketch

of the “first thing the user sees” (i.e., an interface proxy). (b) sketches produced by the programmer for how to organize the content for the game in an XML file that will be input to the game as shown in Figure 27 (i.e., a specification), (c) lists of graphical elements required for the new game design as shown in Figure 28. (i.e., ideas), (d) a tabular action-selector component sketch (i.e., an interface proxy), and two lists that contained rules for the gaming framework.

Figure 27. The developer’s sketch (i.e., a specification) that shows how to organize the content for the game in an XML file.

Figure 28. A list of graphical elements (i.e., ideas) for the new game.

Collectively, the transformative aids included (a) a list of flaws in the game (i.e., bad ideas), (b) a list of the best aspects of the previous sketchbook prototype (i.e., good ideas) that should possibly be reintroduced into the inverted game as shown in Figure 29, (c) numerous stories about the gamer playing the game, (d) sketches for how to organize

the content for the game in an XML file that would become the input to the game (i.e., a specification), (e) a to-do list (i.e., a plan).

Figure 29. A list of the aspects of the Sketchbook prototype (i.e., the software under development) that should be retained (i.e., good ideas).

The team worked on the design of the action selector component, which was required as a consequence of the new game model, work that continued to evolve over the course of subsequent meetings. The first design was a sketch of a list of buttons in a table (i.e., an interface proxy) as shown in Figure 30. No one was satisfied with this design and the lead game designer put the design of the action selector component on the team's to-do list (i.e, a plan). In this meeting there was also an initial attempt at designing the XML content document (i.e., a specification). The programmer proposed two formats, but neither of these seemed to help the content designer structure her work, which I came to appreciate, was very challenging.

Figure 30. The first design for the action selector component (i.e., an interface proxy) is a table with buttons.

Meeting 5, old games and the new game model. In this meeting the content designer called the programmer and the new usability specialist into a meeting and asked them to reflect on the question: What can be learned from previous games (i.e., exemplars) that can help with the design of the new game? The new designer was responsible for the usability aspects of the game and raised a concern about the way previous games had been designed. She did not like the idea a gamer would be able to see a list of characters and actions by hovering their cursor over a character in the game. She said that there was a problem with all previous games from the organization that used this technique (i.e., she provided negative exemplars). The primary tension in the meeting was between the usability of future games, including PlayLearn-it, and the norms for game design at the company.

The meeting was fairly brief. The programmer explained that the behaviour that the usability specialist was concerned about was built into the gaming framework they used to build most games. However, he explained the Sketchbook prototype did not use the gaming framework so that's why hovering over a character could produce something different, like a description, rather than the behaviour found in all other games, which was the appearance of a list of characters and actions.

The outcome of the meeting was a better understanding of what could be designed in the new game. The designers were clearer about what they could design and change.

Meeting 8, the game model is tested with other stages of the game and a clipboard metaphor is proposed for the action selector component. The programmer began this meeting by explaining what he knew about the game and the various stages of the game. He created a list of the stages of the game and the actions that could occur in the stage of the game the team was currently focusing on. The usability specialist, who had requested the meeting, challenged the programmer saying that the Sketchbook prototype (i.e., the software under development) was unusable because any gamer would have no idea how to act. The key contradiction that triggered the meeting was between the game as understood by the programmer and the game as the usability specialist understood it (the usability specialist had not attended the two-hour meeting where the lead game designer explained the inverted game).

The other attendees in the meeting were the game designer and the content designer. The programmer talked about a grid (i.e., a computational proxy), which was a programming construct that kept track of the game's events, i.e., the changes to the game's scores, which occurred when a particular character and action were selected. A reproduction of this grid from my notes is shown in Figure 31. Somehow, this inspired the idea that a useful metaphor for the action selector component could be a clipboard (the clipboard is both a model and a good idea), but this was ridiculed by one meeting attendee who acted out, in a humorous way, how terribly easy it would be to play the game if the gamer used a virtual clipboard. She held an imaginary clipboard and pencil and ticked items off rapidly to make her point (i.e., she told a story).

The meeting concluded with the team developing a deeper understanding of how scores changed in the game and the challenges in designing the action selector component.

Figure 31. Sketch of a grid (i.e., a computational proxy) that maps a person and an action to an event, which identifies a formula that changes the game's score.

Meeting 9, a "tech tree" is proposed for the action-selector component. Meeting 9 was called because the programmer and usability specialist wanted to ask: What is the role of the action-selector component in the game and what is the game? The game designer explained his vision for the game, how he thought about parts of the game, and the reason that current and future actions had to be represented in the game for the gamer. The meeting was called to address the tension between designing a usable action selector component and designing a suitable game for the client.

The other attendees in the meeting were the content designer and the programmer. As the meeting progressed, the game designer suggested the action selector component could be designed liked a tech tree from the commercial game Civilization (which served as an exemplar). The game designer showed the team some tech trees as shown in Figure 32 and showed the team how the idea would work in the game by sketching what the gamer would see as shown in Figure 33.

Figure 32. A tech tree (i.e., an exemplar) from the game of Civilization.

Figure 33. A sketch of what the tech tree would look like in the game (i.e., an interface proxy).

Meeting 10, the action selector component is prototyped. The game designer had been away for a week. On his return the programmer demonstrated a program he had created to implement the tech tree (i.e., an interface proxy), which was not yet integrated into the game, but which was interactive. In this meeting the game designer provided “push back” on the programmer’s idea. He wanted to know if the programmer’s implementation of the tech tree was practical, and if the tech tree would do everything they needed in the game. The meeting was called to address the tension between the

proposal for the tech tree, the overall game design, and the practicality of the programmer's proposal.

During the course of the meeting other nonprogramming options for implementing the action selector component were explored. The content designer joined the meeting at the end and together they decided the tech tree would not be programmed in its entirety, but would be drawn by the artist, and selected parts would be revealed at various points in the game. The programmer would write code that displayed only selected parts of the tech tree.

This analysis of 17 meetings, 6 of which were summarized here, showed how attendees arranged meetings to address important product tensions, and often arrived at a meeting with an artefact that would abet the resolution of the tension. The initial tension was usually a two-way tension, but multi-way tensions occurred occasionally. Typically one, but sometimes more attendees advocated for each "side" of the tension. Inactive meeting attendees were not uncommon. They appeared to be there to deepen their understanding. Within the meetings, attendees initially engaged with the tension that provoked the meeting, but the "agenda" usually broadened to address other important issues. The team used various processes, such as questioning, informing, refining, designing, reflecting, planning, providing feedback, presenting, or demonstrating. These helped to resolve contradictions in the project, which revolved around the development of one aspect of the project given the development of another aspect, misconceptions within in the team, the adoption of an idea from another game, and so on. The resolution of contradictions produced outcomes such as moving the design forward in one or more aspects, improving understanding, reaching agreements on deadlines, making

simplifications, and agreeing on processes. Occasionally meeting attendees transformed artefacts in the meeting (very often a product proxy) as they proceeded. Many examples of successful artefacts used in the transformative process were identified earlier.

The analysis revealed that a good part of the designer's work evolved around developing a common understanding of the game that the individuals within the team could all work with, although this was not an explicitly stated goal of the team. This understanding, that would eventually be built into the software product, was not a static entity. It was a complex of gaming elements and decisions about many aspects of the game, which collectively provided the basic direction for how the game would be played. I showed that the work of both designing PlayLearn-it and the new gaming framework for future games happened slowly over time as the Jaba team deepened their understanding of the game and proposed, assessed, and developed ideas. Sequences of other meetings at Talia, Telespace, and Quadrow, and important artefacts in these meetings are described in Appendix G. The activity observed in these meetings concurred in principle with the activity I observed and described in the Jaba meetings.

Table 30 *Meetings at Jaba*

Mtg #	People	Key artefact	Summary of the meeting	
1	Tool designer, writer, content designer, developer (4)	1st prototype of the game		The writer does not understand the current game. She sets up a meeting to go through a list of questions with the game designer about the first game prototype.
4	Content designer, game designer, developer (3)	Whiteboard content		The game designer critiques the previous prototype and proposes ideas for a new game. The team provides "pushback" and their ideas and questions.
5	Content designer, tool designer, graphic designer, developer (4)	List of things that pop up when hovering, including a sketch of a 'grid'.		The content designer wanted to discuss the lessons learned from old games. Discussion ensues on the hovering event and how it is interpreted differently in different games.
8	Tool designer, graphic designer, artist, developer (4)	Various informal representations of potential gaming actions.	whiteboard sketch of phases, documents and actions. Sketch of a clipboard/grid	The topic is gaming actions. The team is trying to get a sense of what gamers will be doing when they play the game, whether it will be challenging, fun etc. Discussion is also about how actions will be represented in the game.
9	Developer, game designer, then content designer at end (3)	Game prototype		The developer demos a tech tree-like solution for the action selector component to the game designer who provides pushback. He proposes this be integrated into the game. Many issues are raised.

Summary

I found that in each meeting a single major contradiction prompted the meeting. Further, in each meeting, a team member, such as a designer or a developer, typically raised an issue by talking and using artefacts to express a tension or dissatisfaction the team member had with the current prototype. Typically, another team member responded in the meeting with a different verbal message and supporting artefacts explaining their views. The meetings typically addressed this key contradiction, and then developed to address other important matters that came up as a consequence of addressing the key contradiction. Occasionally an artefact was transformed in the meeting. Very often a range of artefacts were used in a meeting.

Meetings had an outcome, which was usually some development of the game, but they also increased the team's understanding of the game. Development of a shared understanding of the game was important. For example, in many Jaba meetings the team searched to agree on, and to understand, the basic game elements like the number of characters, the motif, the game mechanic, and so on. The shared understanding developed in meetings allowed each team member to work on their part of the software creation activity individually.

The framework for the game was a secondary outcome of the game's development. Essentially, the shared understandings about the game's structure eventually evolved into the framework for the game, as the developer incorporated shared understandings into an interactive prototype. This would later become a new gaming framework for a series of future games. This suggests a possible progression from the development of shared understandings within the team (especially between the designers and developer), to project-specific frameworks, to platform-general frameworks.

Study 3: Creative Work within Meetings

This section presents an analysis of design meetings captured on video in which both a designer and a developer took part. The meetings took place in four different organizations. In each one, I tracked a single project and a single designer and developer. I analyzed each meeting in multiple ways to build up a rich understanding of designer-developer interactions in the context of meeting work.

Table 31 provides an overview of the videos that were analyzed in this study. I observed designers and developers working in 18 scheduled meetings, in 5 impromptu meetings, and in 31 chats. The table shows the percentage that were captured on video and analyzed.

Table 31 *Meetings Involving Designers and Developers and Number Video Recorded*

Meeting Type	Talia	Jaba	Telespace	Quadrow	Total
Scheduled	7	3	2	6	18
On video	3	3	0	0	6 (33%)
Impromptu	0	3	1	1	5
On video	0	1	0	1	2 (40%)
Chats	0	13	13	5	31
On video	0	0	4	1	5 (16%)

I begin by presenting a detailed analysis of the longest meeting I attended and video recorded, chosen because it had the most potential for providing repeated patterns of behaviours. Then I review the findings, provide an interpretation, and offer conclusions. I then briefly present the results of the analyses of the other twelve videos and provide further interpretations and conclusions.

This study begins by reporting on an analysis of a video taken at Jaba, an organization that develops educational computer games. I reported this analysis at an

Agile conference (Brown, Biddle, & Lindgaard, 2008a), but at that time had not analyzed any other meetings. As commonly occurs, subsequent analyses of other meetings helped to improve the analysis of the first meeting. For instance, as I analyzed more meetings, the phase names changed (the new names are used in this section), and my coding technique for artefacts moved from being form-focused to purpose-focused. However, because this first analysis revealed interesting results about the form of certain artefacts (e.g. sketches, lists), I have retained the focus on form here.

The meeting analyzed here was a scheduled two-hour meeting of intense design activity. In the video two experienced designers and one developer worked on a game intended to be fun, but also intended to teach something that the client wanted the game players to learn. The game was unusual because it was intended for work environments. In describing this meeting, rather than using the pseudonyms assigned to participants in this research, to protect their identities, I use the names Des1 and Des2 for the designers and Dev for the developer.

Des1 was a gamer and a specialist in game design; Des2 was a learning specialist and a content designer; and Dev, the developer, was knowledgeable about GUI design and was also a gamer; he was responsible for programming the game and integrating content from the designers. Des1 and Dev were “generalizing specialists” (Ambler & Jeffries, 2002; Ambler, 2007) because they both had significant experience outside their specialties.

I used five analysis techniques to analyze the video data and triangulated the results, which complemented each other. I present them here very briefly in the order in which I initiated each analysis. I used activity systems analysis (Engeström, 2000) to describe the design activity in general. This analysis produced a description of the

activity within the meeting. This included a description of artefacts used in the meeting. I used interaction analysis (Jordan & Henderson, 1995), an inter-disciplinary video-analysis technique to look for large patterns in the activity of the meeting, and to look for patterns in beginnings, endings, transitions, temporal orderings, routines, and artefact use. I used grounded theory (Strauss & Corbin, 1998a; 1998d) to analyze the meeting process, and, through this, I identified sequences of goal-oriented, group actions, and then identified the dominant goal in each of the phases from the previous analysis. An analysis of contradictions identified the contradictions or tensions that the designers and developer struggled to resolve, such as balancing the learning aspects of the game with the fun elements of the game to make the game both enjoyable and educational. I carried out this analysis by attending to any talk by the designers and developer that referred to an abstract quality (i.e., cultural objective) of the game they were designing. Finally, I used interaction analysis a second time to analyze five minutes of an interesting part of the video data. This final analysis identified uses of artefacts that designers were not aware of, such as the way they tended to point to aspects of a visual artefact, and say a word at the same time to build up a common activity-specific lexicon amongst themselves.

Jaba Meeting Case Study

The meeting activity. Identifying all the elements of an activity system confirms the existence of an activity, while simultaneously providing a good description of it. While I have previously stressed that software-creation work was a team activity, it is not contradictory to also say that a meeting, a tool/artefact in this broader activity, is also an activity. This simply means that the same human strategies to coordinate work are being applied in two different ways. The first is to create the software generally, and the second is to advance the work in small increments during the course of a meeting. Each activity

is oriented toward transforming separate, but related objects. In the case of the software-creation activity, the object of transformation is the software, and in the case of meetings it is a description or understanding of the software, which is sometimes represented by a software prototype.

In the Jaba design environment Des1, Des 2, and Dev were key players/subjects who regularly met to work on games. Other members of the team were not present and in fact did not usually attend either scheduled or impromptu meetings. These other team members included the usability specialist, the graphic designer, the artist, and the writer. Des1 had been working at Jaba since its inception. His background was in the hard sciences and process engineering, and he was a self-taught gaming specialist. Des2's background was a combination of psychology, learning, and technological skills. She had an M.A. in human-computer interaction. Dev had a background in computer science and had taken one course in user interface design. In this meeting Des1, Des2, and Dev discussed Des1's recent redesign of the game, which he had accomplished over the weekend.

The division of labour within the meeting mainly aligned with the roles within the team. Des1 was responsible for the design of the game mechanics and the profitability of the company. Des2 was the project manager. She also ensured that the content for the game existed and that it was formatted in a way that it could be transformed into input for the game. She was also responsible for the learning aspects of the game. Dev's primary role was as a game developer, but he also functioned as a coordinator of the group's activity in that he integrated gaming rules, content, art, graphics, and so on. Each individual had multiple roles. When the meeting was recorded, Des1 and Dev had

worked together off and on over the previous month to create the first game prototype. Des2 and others were just starting to make their contributions.

The motive of the meeting was to evaluate a second prototype for the new game.

The object of the activity was the gaming prototype as captured by the whiteboard contents, some of the stories, and the designers' and developer's ideas.

A variety of tools were used in this meeting. The designers and developer primarily relied on the whiteboard for recording their good design ideas through use of visual representations, such as sketches of interface proxies, as in Figure 34, and lists of ideas, as in Figure 35. The designers and developer also told numerous stories, raised many questions, and proposed many ideas. They also occasionally referred to exemplars of other software and abstract models. All artefacts used in the meeting are shown in Figure 36.

The group members abided by the normal rules of conversation amongst peers in a North American society. They shared a common visual language for communicating. Talking and drawing happened simultaneously, continuously and naturally throughout the entire two-hour meeting. A tacit rule was that only certain team members took part in this type of meeting. The graphic designer, artist, and student programmer generally did not attend.

Figure 34. A sketch that captured a designer's idea about game actors.

Figure 35. A list that captured a designer's idea about game icons.

Phases in the meeting's activity. I used interaction analysis to identify phases in the meeting activity, despite the lack of obvious structure in this agenda-less meeting. To conduct this analysis I transcribed the video and annotated it with screen shots of artefact use. Then I scanned the annotated text for instances of artefact use and highlighted these. From this I created a list of all the artefacts that were used ordered by time. If an artefact was re-used it went into the list more than once. The list included all instances of the usage of whiteboard sketches, lists, stories and so on. Once the list was compiled I divided it into large chunks of related artefacts. I double-checked with my text to make sure the chunking made sense whether looking at the transcribed text or the artefact list. This resulted in a little fine-tuning of the beginning and ends of my chunks.

The process of identifying common artefact sets and grouping these, shown pictorially in Figure 37, resulted in the identification of 12 meeting chunks, which I named phases. On average, a phase lasted roughly 10 minutes (120 minutes divided by 12 chunks). Next I describe the purpose of the first and second phases.

Phase 1 began with a presentation by Des1. Ten artefacts were created during the initial phase of the meeting. One was a list of flaws (i.e., bad ideas), the others were

sketches (5) and stories (4) that related to the action of hovering over a character in the game, entering a game phase, and then clicking on that character. Together these artefacts showed a high-level vision of the new prototype, so I grouped them together.

Phase 2 began when Des1's presentation ended and there was an increase in the participation of Des2 and Dev. The next artefacts, the stories (all 29) and sketches (4 of them) were grouped together because the designers and developer used them to jointly discuss and create artefacts that related to what happens when you click on a game character and then subsequently select an action. Relative to the initial presentation, in phase 2 the designers and developer explored the new design more deeply by extending their consideration of the potential game player's actions, hence the artefacts used to do this were chunked together.

In phase 2 the interaction that was explored in great detail was simply hovering on a character and clicking (i.e., selecting it), but by phase 8 and phase 11, the designers and developer were considering issues that related to an entire game, such as how a player would comprehend the game state and how they would switch between stages of the game. The following summary shows how the phases that explored the interaction of future players with the game were organized. The designers and developer progressed through a consideration of gaming actions that they themselves suggested through stories, which collectively re-enacted a kind of patchwork of game plays from first hover over a character to the point where gamers were able to navigate through the four stages of the game. While the stories contained different main characters, different contexts, and differed in detail, collectively the stories covered a re-enactment of the game from the beginning of the game to a very advanced state in the game. This work of progressing

through the game bit by bit happened in several phases and was interrupted periodically by another type of work they did, which was evaluative work I have called “assessing”.

Below, I briefly describe the phases relating to the re-enactment of a game play. Designers and developers addressed only portions of the game play at a time. These progressed from actions that happen at the beginning of the game, such as selecting a character, to actions that happen mid-game, such as considering one’s strategy, to actions that happen at the very end of a game, such as reflecting on one’s gaming performance.

- Phase 2 gamer actions (gamer actions are potential moves made in the game by an imagined gamer) considered by the team included selecting a character, and then selecting the create action.
- Phase 4 gamer actions considered by the team included viewing a chronology, consequences of selecting an action, setting multiple characters to act in one turn, and selecting an action.
- Phase 6 gamer actions considered by the team included selecting an action, and allocating a task to another character or a second character.
- Phase 8 gamer actions considered by the team included selecting a series of actions, players playing ineffectively, and the feedback mechanisms players would experience.
- Phase 11 gamer actions considered by the team included navigating the game space to play multiple stages of the game.

The whiteboard was used as a repository for good ideas. Twenty-six visual artefacts were identified on the whiteboard, which were created throughout the course of the meeting. The stories (124 of these) and the visual artefacts (26) together were used to

support an exploration of Des1's ideas. Throughout the meeting, 41 ideas were proposed and 10 challenging questions were posed. Meeting artefacts are shown visually in Figure 36.

This analysis shows how many artefacts are used in meetings, and especially the variety of artefacts that often appeared in the designers' and developer's talk, such as stories of gamer actions, models such as Bloom's taxonomy, game types such as rock-paper-scissors, data models, or exemplars such as tech trees from the game Civilization. Ideas were expected artefacts, but questions, unexpected artefacts, revealed themselves to be important in the work of creating software.

Through use of a modified version of interaction analysis I found that there was order in the ebb and flow of the work, and that artefacts were crucial for focusing and abetting the progress of the activity. Details relating to the phases and the artefacts used in each phase are found in Table 32, which also contains details relevant to other analyses of this video.

Icon	Artefact Type	Examples	Occurrences
Model	Rock-Paper-Scissors (a gaming construction metaphor)	2
Questions	You're teaching them to tie actions to employees, is that right?	10
Ideas	"I think an excellent idea would be to delete ... to let them not have all the actors. Remove the facilitator from game."	41
Exemplars	"...how logs work in Bolder's Gate and Never Winter Nights"	7
Sketches		20
Lists		6
Stories	"You go to an actor. Let's say this actor is the DQ. You go to the actor and you hover over the actor – a little cursor's over the actor – the text box popup pops up, the text box will give a description about that actor and its roles."	124

Figure 36. Two hundred and nine artefacts used in the Jaba meeting by category.

Phase	Artefacts			
Simulating
Assessing
Simulating

Figure 37. Each phase is centered around a set of artefacts. The artefact types within a phase are similar types for phases of the same type.

Table 32 *Patterns in Talk- and Artefact-Use in a Design Meeting at Jaba*

Legend—S Stories, V Visuals, T Total, Des1 Game designer, Des2 Content designer, Dev developer, Q questions, I Ideas

Phase	# Artefacts	Visuals	Stories about	Major contradiction between	I	Q
1 Des1 presenting (1,500 words)	S=4 V=5 T=9	1 List flaws in current game 2 Sketch first screen of phase n 3 Sketch popup menu	1 Hovering over the DQ character 2 Entering game phase n 3 Clicking on the DQ character	Gameiness versus Workload vs Usability	8	0
2 Simulate-1 (3,300 words)	S=29 V=4 T=33	1 Reuse sketch of first screen 2 Reuse sketch of popup menu 3 Sketch popup menu in another phase	1 Clicking on a character 2 Selecting an action	Gameiness vs Complexity of the game	5	3
3 Assessing the previous and current games (1,500 words)	S=10 V=1 T=11	List good aspects of the previous game	1 Dissatisfied players 2 Satisfying aspects of previous game	Gameiness vs how well the game supports learning	0	5
4 Simulate-2 (2,100 words)	S=17 V=2 T=19	1 Reuse sketch of the first screen 2 Reuse sketch of popup menu	1 Viewing game chronologies 2 Selecting an action 3 An "I'm done with this turn" button 4 Making character busy	Gameiness vs Complexity of the game vs Authenticity	7	0
5 Assessing the game's requirements (600 words)	S=8 V=0 T=8		1 Other games 2 Learning outcomes	Fun factor vs Client requirements vs Supports learning	4	0
6 Simulate-3 (2,400 words)	S=22 V=4 T=26	1 Reuse whiteboard sketch of first screen 2 Reuse sketch of popup menu	1 Selecting an action 2 Interrupting characters 3 Characters sharing tasks	Complexity of the game vs Fun factor vs Supports learning	7	0
7 Assessing the effectiveness variable (200 words)	S=2 V=0 T=2		1 Players lacking awareness of effectiveness score	Gameiness vs Client requirements	0	1
8 Simulate-4 (2,200 words)	S=19 V=3 T=22	1 Reuse sketch of first screen 2 Reuse sketch of popup menu	1 Player feedback for ineffective choices	Amount of feedback to the user vs Playability	5	0
9 Designing the format for the game content (800 words)	S=2 V=6 T=8	1 Two sketches organization of gaming content	1 The production of content for the game	Ease of implementation vs Intuitiveness of the content layout	2	0
10 Assessing the new game's objective (400 words)	S=1 V=1 T=2	1 Reference to a design document that should be updated	1 Players engaging with experience logs	Workload vs Gameiness vs Fun factor	2	0
11 Simulate -5 the interface to simplify the work of the artist (1,000 words)	S=10 V=0 T=10		1 Navigating large floor plans	Client requirements vs Workload vs Authenticity	3	0
12 Next Steps (200 words)				Office morale vs Production of the game	0	0

Dominant actions within each meeting phase. I used grounded theory to look more closely at the meaning behind the artefact-mediated interactions of the designers and developers. This allowed for the determination of the dominant action of each phase.

To conduct this analysis, I examined every jointly-achieved action in the two-hour video and then determined the dominant action in each phase. For example, in phase one, the most frequently observed joint action was presenting an idea. Although Des1 was the presenter, this action was also a joint action because Des2 and Dev were participants in it. Therefore the dominant joint action of phase 1 was Des1 presenting ideas for a redesigned game. The dominant joint action of each of the 11 phases and the key artefacts used during those phases were

1. Presenting ideas for a redesigned game: Des1 used two graphics to illustrate what he called the new and “inverted game” to Des2 and Dev. The stories were about basic game player interactions such as “what happens when the player clicks on a particular character.”
2. Simulating a gamer playing the game (1st pass): Des1, Des2, and Dev used any stories, which I call “game play” stories to make the description of the player’s interactions more real. The game play stories in this phase simulated the player’s initial actions in the game. Other stories were “worry” stories, which expressed concerns of the story teller. “Idea” stories presented, tested or explored ideas that the designers and developer raised.
3. Assessing the previous and current gaming prototypes: This phase began with a worry story, which was about the game being too boring. Then a list of good aspects of the previous game prototype helped to identify some aspects of the game that could be carried forward. “Visionary” stories in this phase were about

the fun aspects of other Jaba games, and how these aspects might be applied to the new game.

4. Simulating a gamer playing the game (2nd pass): The stories in this phase were much like the stories in the 1st redesign phase in nature. The key difference was that the stories helped to advance the simulation of the new game play so that more aspects of the game play were considered.
5. Assessing the game requirements: This phase saw a combination of worry stories and visionary stories, much like the type of stories found in the previous assessing phase.
6. Simulating a gamer playing the game (3rd pass): Similar to Phase 4.
7. Assessing the point of the effectiveness variable: Similar to Phase 5.
8. Simulating a gamer playing the game (4th pass): Similar to Phase 4.
9. Designing the format for the game content: This phase was unique in that it was about how to get “content” into the game in an efficient way. This involved designing the format for a Word document in such a way that software could be written to extract the content of the Word file and produce a format of the content that the gaming engine could use as input to the game.
10. Assessing the new game’s objective: Same as phase 5.
11. Simulating a gamer playing the game to explore the artwork required for the game (5th pass): Similar to Phase 4.
12. Next Steps: Here the group addressed what they should focus on in the immediate future. No stories or visual artefacts were created in this phase.

Aside from the opening and closing phases, throughout the meeting the designers and developer alternated between simulating the game and assessing the project, which

primarily addressed the business reasons for the game. Figure 38 shows a graph highlighting how the number of words and artefacts varied from phase to phase. Simulating phases, relative to all other phases, generated more ideas and more talk. Simulating phases also involved many more artefacts in total, and of these, the majority of them were stories.

The analysis further revealed that Simulating phases had two parts. In the first part, called Proposing a New Idea, a designer or developer would propose an idea that related to the interaction being considered at the time, such as the player selecting an action. The idea was sometimes conveyed through a drawing, but more frequently by telling a story. In Proposing subphases, ideas were frequently presented through stories, and one story elicited others; collectively, these stories served to clarify and develop ideas. The second part of the Simulating phase was Evaluating an Idea. The nature of this phase was revealed in the subsequent contradiction analysis, as was the nature of the Assessing the Project phase. Figure 39 is a visual representation of the software-creation activity showing paired phases and subphases. Arrows show flow of time. The entry point, which was Des1's presentation, is not shown because there was only one entry point in this analysis, and therefore insufficient data for a generalization. The exit point, also not shown in this diagram for the same reason, was typically the construction of a to-do list. A third subphase, called *Checking the Implementation Status*, was discovered after analyzing another 12 meetings. This subphase is shown in italics. In this phase designers would check with developers about the status of the implementation of the product.

Types: C is simulating, R is assessing, Oth is Other phases

Figure 38. Counts of words, stories and visuals for various phases.

Figure 39. Phases and subphases in meetings.

Figure 40. Proposing and Evaluating subphases and Assessing phases ordered temporally.

Tensions within the meeting. This analysis was conducted by examining the talk of the designers and developer, a method that revealed conscious contradictions or tensions in the activity. Within this one meeting, the designers and developer had many aspects to balance against each other including: “gameiness” (i.e., the experience of the ludic elements in a game by the end user), workload, usability, complexity of the game, how well the game supports learning, authenticity, the game’s fun factor, client requirements, user abilities and needs, playability of the game, the “integratability” of the designers’ work, and finally “implementability.” These 12 issues or concerns acted like forces that pulled the game in one direction or other were identified largely from the talk of the designers and developer, so the terms are theirs and not mine, and are depicted in Figure 41 as a multi-way contradiction in the project work. Next I show how this multi-way contradiction was addressed in each of the phases.

The first prototype of the game was developed by Des1, the lead game designer, and Dev, the developer. The game designer was aware of the content that must be taught

and his challenge was to design a game meeting that requirement. However, the greatest concern for the first prototype was that “there was a game.”

In Table 32, in phase 1, Des1 began by saying that the game was “too confusing,” which was interpreted in this analysis as a reference to the usability of the game. His second concern was workload, and his aim was “less work for everyone.” Therefore, one would expect the new design, expressed through his whiteboard sketch, would still be “gamey,” but would also be more usable and less work to build. His sketches exposed his ideas to others, and allowed them to judge whether or not this was so. It also allowed them to comment on other aspects of the game design. In all, seven utterances revealed contradictions in this phase; the 3-way contradiction between gameiness, workload, and usability experienced by the team, which was revealed through their utterances, in this phase is summarized in the ‘Contradictions Between’ column of Table 32.

In phase 2, twelve utterances revealed contradictions. As the team explored Des1’s ideas other contradictions emerged. The key contradiction in phase 2 was complexity. As Des1 said, “What we’ve got is a simple matching game and that’s fine ... what I want to do is ... to add ... complexity,” or as Des2 puts it “so as a player you might think, this is too easy—[but] then you’ll learn quickly that you’re not optimizing your variables.” Four other elements to be balanced were also mentioned in this phase.

In phase 3, the dominant contradiction was “gameiness” versus “how well the game supports learning.” As Des1 puts it “We also have to deal with a learning issue ... [because if you see all of the actions] you will want to do them all.”

In phase 4, two concerns dominated. The first was complexity, or as Des1 says, “But I’m concerned, along with Dev that that’s a bit complicated.” The second was authenticity as indicated by Dev’s brief comment, “That’s not real right?”, referring to

the possibility that an aspect of the game world was imagined rather than a real part of the authentic world they were simulating.

Phase 5 saw the group address the “fun factor” of the game as well as client requirements and how well the game supported learning. Phase 6 addressed multiple factors, and phase 7 returned to client requirements. Phase 8 addressed the issue of a player playing an entire game. The ability of the game to support learning in general was assessed alongside “playability.” Phase 9 dealt with the specific issue of content design, and the tensions were more of the type a developer might regularly experience, i.e., organization of data versus ease of implementation. In Phase 10 the designers and developer evaluated issues to do with the content, and the concern was mainly about workload. Phase 11 was again about workload, but this time to do with the artistic requirements of the game (i.e., drawing the characters, the scenes and objects). In phase 12 the only contradiction raised was between the Jaba PlayLearn-it team’s morale as a consequence of the required redesign and re-production of the game.

This analysis suggests the reasons for the production of the stories and visual artefacts. It shows that the designers and developer were attempting to resolve a complex multi-way contradiction within one meeting (I counted 12 software attributes and project goals that the trio consciously addressed). However, these were not simultaneously addressed. Over brief periods of time in the meeting anywhere from one to three, and very occasionally, four attributes or goals were considered simultaneously, but never more. With regard to this, the representations and stories, because of their richness, allowed the designers and developer to assess multiple aspects of the game at once.

However, there were some patterns to do with the manner in which contradictory attributes were addressed. When evaluating ideas, the contradictory attributes that were

addressed tended to be responses to the immediate idea being proposed (i.e., the consequences of the idea for impacting gameiness, complexity, authenticity, the game's fun factor, how well the idea supports learning, implementability, and playability). In contrast, the contradictory goals addressed in the phases where the project was assessed, tended to address larger issues such as client requirements, user needs, or workload implications. The rows in Table 32 corresponding to Assessing the Project phases (rows 3, 5, 7 and 10) provide evidence of this pattern.

Nonconscious uses of Artefacts. The designers and developer in this meeting made heavy use of the whiteboard during certain phases and told numerous stories. For the designers and developer, the obvious and conscious use for the sketches was to represent the user interface and to draw attention to good design ideas. The obvious and conscious use for the stories was to raise an issue about usage of the software. The purpose of this analysis was to find additional uses of the artefacts that were not conscious uses. This was achieved by using interaction analysis to micro analyze every artefact-mediated operation in an interesting, five-minute segment of the video (roughly speaking, a conscious action breaks down into smaller movements called operations). The outcome of this analysis was two-fold. The first was a very detailed description of the designers' and developer's actions as they worked. The second was lists of ways that certain artefacts were used at a nonconscious level by the team. The following is a condensed version of the detailed description.

The video segment selected for analysis was the first five minutes of phase 2.

Des1 had just finished presenting his ideas to Des2 and Dev. He had changed his vision

Figure 41. Results of analysis of contradictions, a 12-way contradiction. The team aimed to achieve all 12 attributes (i.e., objectives) within the one game; the challenge for the team was to work out how to do this. At times when attributes were in conflict, occasional tradeoffs were necessary. The visualization shows the many attributes that required consideration. Some, but not all, were in conflict with each other.

for the game, and this was the first time he was expressing his ideas on a whiteboard. The change was a fairly drastic one. In Des1's words, the game had been "inverted."

The video segment began with Des2 asking Des1 if his representation in a previous sketch purposely excluded two actions she believed had been omitted. Des1 tells her that they're meant to be there, but he just did not write in all of the actions. This suggests that a sketch is a sketchpad of partially formed ideas that serves as a prompt to others who may wish to respond and develop them. These comments also established a shared understanding that the representation was incomplete, and aided in the interpretation of the visual that was acting as a representation of the group's shared understanding.

Des1 then explained that what had been called activities in the old game had become actions in the new game pointing to the action icons in the representation. He emphasized the word “actions” and pointed to the action icons at the same time, thereby developing, or likely re-enforcing, the meaning of the icons. Des1 then went on to explain that there were three types of icons and he made a list, which was a way to group related ideas that he wished to address and needed to recall later. He, Des2 and Dev corrected this list until it was just right and the meaning of each icon was understood, further enforcing the idea that a visual representation helps to develop shared meaning.

The rich description that is the output of the micro-analysis continued in this way, analyzing every operation of the designers and developer that involved the use of artefacts. I found that sketches were for focusing attention, developing a set of named symbols to be used in later conversations, recording design decisions, exploring relationships between design components, and making to-do notes of design issues still to be addressed. Stories were for raising problems, defining scope, articulating typical requirements, exploring boundary conditions, relating designs to particular users and user objectives, and keeping the design activity interesting and fun. These were all useful operations when simulating the game in use because they challenged the current understanding of the game in many ways and allowed the game to develop further.

Stories and sketches together were powerful and had a different purpose than each artefact individually. The designers and the developer acted out stories using sketches, which revealed errors in the game, or temporal information such as task sequence, or contextual information; they also served to reveal contradictions between the visions of the system held by Des1, Dev2 and Dev.

This analysis served to identify the many ways that artefacts supported the design activity and showed clearly that the role of artefacts extended beyond representing the design of the user interface or, in the case of stories raising an issue through narrative means. Detailed results are found in Table 33 through Table 36.

Table 33 *Value of Sketches in Interactions Below Level of Conscious Awareness*

Area of Value	Value
Communication	<ul style="list-style-type: none"> • Supports economy of expression • Affords interaction • Limits interaction • Reflects the drawer's values (e.g., maintaining consistency) • Represents shared meaning • Supports the development of a set of iconic named symbols • Can be used to indicate a part of an interface • Reflects the relative position of elements on an interface
Memory	<ul style="list-style-type: none"> • Records design decisions being made • Records issues addressed (and by implication not addressed) • Reflects the depth to which design issues have been addressed
Exploration	<ul style="list-style-type: none"> • Expresses partially formed ideas that others may respond to and develop • Acts as a stage upon which stories can be enacted • Serves to reveal contradictions/tensions, especially between disciplines • Serves to reveal inconsistencies
Organization of the Work	<ul style="list-style-type: none"> • Keeps the work focused • Can set the topic

Table 34 *Roles of Stories in Interactions Below the Level of Conscious Awareness*

Area of Value	Value
Communication	<ul style="list-style-type: none"> • Develops a shared understanding of the project requirements • Reminds others of project requirements
Exploration	<ul style="list-style-type: none"> • A means for proposing a design solution • Reveals future problems • Uncovers project requirements • Explores ease of use • Explores time to execute a task • Checks that functionality is present • Checks the emerging interface against reality
Motivation	<ul style="list-style-type: none"> • Keeps the work interesting • Keeps the work fun • Keeps the work personal • Keeps the work focused on significant end user problems

Table 35 *Roles of Stories and Sketches Together in Interactions*

Area of Value	Value
Exploration	<ul style="list-style-type: none"> • Revealing temporal information such as task sequence • Revealing contextual information • Revealing errors • Revealing contradictions between what is depicted visually and what is said. Ideally, the visual and oral act in a complementary way to confirm each other. If this doesn't happen, a contradiction is revealed

Table 36 *Role of Lists in Interactions*

Area of Value	Value
Memory	<ul style="list-style-type: none"> • Forms a group of related item that need to be remembered
Organizational	<ul style="list-style-type: none"> • A means for ordering the activity (e.g., an agenda)

Interpretation and Significance of the Jaba Case Study

A goal of all the analytical techniques used in this part of the research was to emphasize the role of artefacts. The results of the analysis revealed something about the nature of designer-developer collaborations and the nature of the artefacts used when designing that is valuable information for those working in design environments.

The activity systems analysis produced a good description of the activity within a particular design meeting. The motive of the meeting was to develop a vision for the next game prototype. The analysis identified the artefacts and the collaborators. It revealed that the primary tools in the design meeting were the whiteboard sketches, lists, design stories, and questions, but that the designers and developer also used models and exemplars. The object of the activity was the gaming prototype as captured by the whiteboard contents, some of the stories, and the designers' and developer's good ideas. This analysis also described the roles of the various collaborators in the design meeting. The analysis showed how the collaborators were collectively responsible for evolving the vision of the game, although each one brought unique skill sets and different concerns.

Interaction analysis revealed the activity of the designers and developer could be segmented into phases, with each phase oriented around a different set of artefacts. A surprising result of compiling a list of all the artefacts was the sheer number of artefacts and the strong reliance on designer stories. While it appears that sketches play a dominant role, this analysis makes clear that stories and questions were also key to the design activity in this meeting. The analysis also showed how some phases in the activity were unique in that they were oriented around considerations of the game player's actions (hovering, clicking etc.). However, it also showed that considering player actions was not

all that designers and developers do in design meetings. Some phases did not consider player actions at all.

Grounded theory revealed the primary purpose of the phases (opening the meeting, presenting, simulating the game, assessing the project, and closing the meeting). While this meeting had opening and closing phases, this analysis showed that the bulk of the meeting alternated between Simulating and Assessing phases. The Simulating phase was associated with more talk, more story telling, and more ideas relative to other phases, suggesting strongly that story telling is crucial to working with ideas, one of the dominant actions in this phase.

Simulating phases have two subparts. In the first part, team members proposed an idea, in the second part they evaluated the idea against various criteria. This cycle was repeated a few times until for various reasons, the team switched to a different phase and began to assess the project in its entirety. Assessing the Project was seen as a separate phase because this activity was oriented around a completely different set of artefacts than the Simulating phase. In the Assessing phase, the team effectively re-pointed themselves in the right direction, or in many cases, simply assured themselves that they were creating the right product.

I also looked closely at stories, because they dominated, and found that stories played multitudinous roles in design activity. Game play stories simulated the play of the game, worry stories raised a concern, idea stories served to convey an idea, and so on. This study shows that certain types of stories occurred regularly in the meeting. These stories were detailed, concrete, and lengthy.

The contradiction analysis revealed reasons for the development in the design activity, namely the specific contradictions or tensions that the designers and developer

were working toward resolving. One result of this analysis was a revelation of the nature of the Evaluating Idea subphases. Evaluating Idea subphases were often a time when the product's attributes such as usability, playability, "gameiness," and so on, were considered in relation to one another. The resolution of these contradictions was the primary purpose of the sketches, lists, and stories that were used during either of these phases. In contrast, I found Assessing the Project phases comprised contradictions of a different nature, which addressed what product ought to be developed when balancing the needs of the client, user, and the team itself against one another. These contradictions were more likely to be balanced through talk alone.

Finally, interaction analysis at the operation level revealed numerous nonconscious uses of artefacts. In essence, at this level of analysis, the artefacts served to extend the designers' and developer's psychological abilities such as remembering, self-regulating (i.e., ordering their behavior), enhancing focus, making associations, judging, and so on. All of this contributed to the ability of the designers and developer to collaborate more effectively and to develop their joint vision of the game.

This analysis also showed it was important to consider the role of artefacts that were jointly used. For instance, I found that the designers and developer used sketches and stories in combination because it helped to reveal errors in a design. In other words, it helped to show what it would be like for the protagonist in the story to play the game. This analysis suggests that artefacts in combination are more powerful than artefacts individually.

A detailed study of collaboration between designers and developers, with an emphasis on the role of artefacts was presented. The study simultaneously analyzed talk- and artefact-mediated actions at various levels of analysis including the activity level, the

level of goal-oriented actions, and the operation level. Key features of this analytical approach were a careful choice of unit of analysis, and conducting complementary analyses selected on the bases of the purpose of the study.

The results suggest a coherent set of interpretations. The main activity analysis confirmed the key role of artefacts, and the interaction analysis showed that this collaborative meeting progressed in phases. The grounded theory analysis showed that the phases primarily alternated between Simulating and Assessing phases, which were distinguishable by the artefacts used in the phase. Contradiction analysis revealed how forces reflecting different objectives for the game, and an emerging understanding of the business reasons for the game were at play in the meeting. Finally, the operational level analysis showed how the artefacts extended psychological processes.

This analysis concentrated on one collaborative meeting. As such, of course, caution must be exercised in generalizing the conclusions. The next section shows how the models presented in this section were extended in the analyses of 12 other videos.

Analysis of Meeting Phases at Jaba, Talia, Quadrow, and Telespace

Twelve other video recorded meetings were analyzed in the same manner as the Jaba meeting above, to identify the phases and the artefacts used in these phases. Details of these meetings including attendees, a brief description of the meeting's purpose, the length of the meeting, the type of meeting, and the order of the phases in the meeting are found in Appendix H. In this appendix, unusual phases have been underlined to highlight them, since they were not included in the meeting model. The number of common phases I found in each meeting is shown in Table 37.

Table 37 *Counts of Key Meeting Phases by Meetings*

Meeting	Mins.	Presenting	Enacting	Exploring	Simulating	Assessing	Other
Jaba A	120	1			5	4	3
Jaba B	60		1	1	7	3	4
Jaba C	30	1		2	1		4
Jaba D	5						3
Quadrow A	90		2		8	3	3
Quadrow B	15			1	1	1	2
Telespace A	20			1	1		2
Telespace B	5		1		1		2
Telespace C	10		1		1		2
Telespace D	5				1		2
Talia A	45		3				2
Talia B	60	10			1	7	2
Talia C	60	9			2	5	3
Total		21	8	5	29	23	34

Each of these meetings also had one opening phase and at least one closing phase, explaining 26 of the 34 items in the Other category. One of these phases was explained by one team closing their meeting twice, because the first closing triggered more work. The other eight phases were unusual phases. Their types were Modelling, Reviewing, Considering Implementability, and Planning. I omitted these rare phases from the table, and discuss them only briefly, because I know much less about them.

The dominant way of identifying phases was by reviewing meeting transcripts and identifying the points in the transcript when the meeting activity shifted between one set of related artefacts and another. In my understanding this corresponded to the end of one phase and the beginning of another in the meeting. In naming the phase type, the dominant joint artefact-mediated action in each phase was considered to indicate the

phase type. For example, if most interactions were simulating actions then the phase was considered to be a simulating phase. Therefore, any phase could contain brief actions that were the dominant action of another phase. Interestingly, throughout most phases, evaluating *subphases* were ubiquitous, suggesting that the resolution of product contradictions was an omnipresent concern within the meetings. Designers and developers were always making assessments relative to important product objectives. For example, in the analysis of the Jaba meeting I showed in detail how ideas were constantly being evaluated in Simulating phases against objectives such as gameiness and usability and “the fun factor of the game.” While short Evaluating subphases were ubiquitous in all phases, there were also longer Assessing phases that were qualitatively different in nature. In Assessing phases attendees considered their work from a client- or business- perspective and balanced the needs of the client, user and the team itself against one another. These longer Assessment phases occurred regularly throughout the meeting activity as can be seen in Table 32.

Four kinds of phases were described in the last section. These were (a) Opening: settling in; (b) Closing: wrapping up and planning future actions; (c) Simulating: simulating the use of a mock-up or prototype; and (d) Assessing the project: considering the work from a client- or business- perspective.

As a result of further analyses, additional phases were identified. These were (e) Presenting: one team member describes their ideas for a period of time, with minimal interruption or comments from the rest of the team; (f) Exploring: the team considers multiple options for a design; (g) Enacting: the team works at developing a dynamic understanding of the context in which their software will be introduced by enacting tasks in that real context or a simulated context.

Other phases were (h) Considering Implementability: exploring the feasibility of a design with regard to its ability to be easily implemented. (i) Planning: ordering the work to be done. This work was normally a part of closing the meeting, but once I saw this happening in the middle of a meeting. (j) Reviewing: one reviewing phase occurred in a meeting that was dominated by Simulating phases. In this phase the lead designer reviewed the meeting up to the point they were at in the meeting. (k) Modelling: in this interesting phase, which only occurred once, a design team carefully considered and chose a mental model for their work from several that they reviewed. Usually teams did not consciously select and discuss the mental models they would use in their work.

A Model of Meeting Activity

The purpose of analyzing other video recorded meetings was to further develop the model of concept design meetings involving designers and developers. In developing this model, there was no need to distinguish between scheduled meetings, impromptu meetings or chats, and therefore no need to develop separate models for each meeting type. Seven meeting phases were identified; however, not all phases were present in every meeting. Meeting participants order the Assessing, Presenting, Simulating, Exploring, and Enacting phases as appropriate to their purposes at any point in the meeting. There was generally only a vague plan for any meeting, and certainly no particular plan to engage in any of these phases. Phases began as the need arose in the course of the work. Figure 42 distinguishes four phases that transformed the object of the meeting activity directly. These were Presenting, Simulating, Exploring, and Enacting. Figure 43 presents the same model, but includes some temporal information about the relationships between the phases. Meetings opened in various ways, all of which helped attendees settle in. Phases within the circle occurred in any order. Meetings wound down

and then occasionally resumed. The line dividing the circle into two identifies two groups of phases. One set of phases was about managing the project and the other was about creating the software.

In Figure 44 the meeting model has been enhanced to link phases with artefact types. This model shows that some tension within the product (e.g., the simple fact it is not complete or the absence of a desired project attribute such as usability) triggers a meeting. Phases rely on certain artefact types as indicated in the diagram. These can serve to organize a phase, support the interactions of the phase, or are the primary outcomes of a phase. When a meeting closes, plans are typically made to ensure the team pursues good ideas. However, the main outcome of most meetings is a greater understanding within the team of the software product they are aiming to collectively create.

Figure 42. Phases in meetings involving designers and developers. The transformational work of developing the object of the activity occurs in presenting, simulating, exploring, and enacting phases.

Figure 43. A model for the activity in meetings involving designers and developers. Seven meeting phases are shown. There is no ordering of the phases within the circle. Meetings may wind down and resume. Phases on the left side of the circle are project-focused, those on the right are focused on the interface object.

Figure 44. A model for meetings of designers and developers that shows the typical types of artefacts used in various phases.

Summary

Software creation is a process involving multidisciplinary collaborations. These analyses have focused on developing a greater understanding of meetings that involve designers and developers. The analysis of 13 meetings identified phases common to these meetings. Separate phases existed for opening and closing the meeting. Opening the

meeting involved setting up and settling in. Closing the meeting frequently involved the creation of to-do lists and planning. Neither of these phases relied heavily on artefacts of any sort, except for lists. Another phase that did not rely heavily on explicit artefacts was assessing the project. This analysis found that this was a regularly occurring phase in many design meetings. While these phases were important, the bulk of the work transforming the object of the meeting, which was usually a prototype of the software being created, happened in four different types of phases: Presenting, Simulating, Exploring and Enacting. These were described in the previous section. All of these phases relied heavily on artefacts.

A wide variety of artefacts were observed in use. Stories in meeting were a frequent occurrence. They captured many elements of design work, including user descriptions, context of use, interaction sequences (corporal and temporal), exceptional cases, and so on. Models and exemplars set the direction for the software. Text-based documents and static representations (e.g., sketches) were of limited value for designing some products (e.g., games). Stories used in conjunction with sketches were shown to be valuable. Lists helped designers and developers to organize their work and make distinctions. Most artefacts were transitory and most useful at the time of their construction.

Overall Discussion and Conclusions

This chapter a) reviews the quality of the research conducted in general, b) critically assesses each of the four parts of this research, c) integrates the findings of the four parts, and d) provides some concluding statements and ideas for future research.

Criteria for Constructivist, Qualitative Research and an Evaluation

Morrow (2005) described overall criteria for qualitative research and specific criteria for various types of qualitative research, including constructivist research. These criteria are used here to assess this research against the research paradigm and the philosophy of science within which it was conducted. Morrow's criteria reflect standards for the trustworthiness of the research. Each of her criteria are used to evaluate this research below.

Social Validity

Software creation activity is a socially valid domain of inquiry due to the ubiquity of software in modern society. Further, software failure is not an unusual occurrence, and much human frustration and wasted time, money, and effort can be attributed to deficiencies in various parts of the systems that produce software, including poor software management and poor software design (Flowers, 1996; Glass, 1999). In addition, well-designed software products can support, enhance, and even save lives (Owen, Béguin, & Wackers, 2009). Therefore it is important to understand how software is created. This research contributes toward understanding software creation activity from the perspective of the designers and developers whose artefact-mediated actions and interactions directly result in the creation of the software.

Subjectivity and Reflexivity

The point of managing subjectivity (i.e., addressing the issue of the objectiveness of qualitative research) and reflexivity (i.e., demonstrating that a questioning stance was maintained throughout the research) is to achieve fairness (i.e., effectively soliciting and honouring different constructions of a reality equitably), which makes research more trustworthy.

One important indicator that fairness is achieved is to acknowledge that any research is necessarily conducted in a subjective way and to directly address this subjectivity. To this end, I included a description of my personal background. To ensure reflexivity (i.e., that I continuously reflect and question not only my data, but my subjectivity), I kept a journal, wrote memos, constructed intermediary analysis artefacts to aid with reflection, and applied qualitative methods conscientiously.

Another way to achieve fairness is to design the research so that multiple constructions of the phenomena of interest, in this case, designers and developers engaged in software creation activity, are solicited. To achieve this aim I focused the research on a single concrete project in each organization and interviewed a minimum of two, and as many as four, individuals engaged in the identified project work. Further, because a single case study could not fairly represent the activity of designers and developers generally, multiple projects were studied in different organizations so that a number of designers, developers, and their managers (who were also designers and developers) could express how they experienced software creation work.

To further achieve fairness and ensure the research was not biased toward one category of participant over another, designer and developer issues were evenly

addressed throughout the analysis and writing, avoiding a lopsided interpretation of the data.

All of these practices helped me to interpret the data more truthfully and to represent my participants' realities fairly.

Adequacy of the Data

Morrow (2005) also identifies adequacy of the data as an important criterion for good qualitative research. For these studies, the purpose of the data collection was to ensure a deep meaningful understanding of my participants and their activities. Morrow identifies a number of specific criteria relevant to assessing the adequacy of qualitative data.

In this research, maximum variation of data was sought. In particular, recruitment methods deliberately aimed to sample a diversity of designer and developer types, design and development approaches, and software products. I achieved this aim by recruiting individuals, organizations, and projects that were different from others I had already recruited. However, to enter any organization I had to have the backing of three to five people and sometimes also security clearances and nondisclosure agreements.

Consequently, I suspect that my research is slanted toward participants and organizations that follow better software practices and projects that do not represent a high security risk, i.e., organizations that were more open. Further, I required that all designers and developers in this study worked face-to-face because direct, and in many cases prolonged, engagement with my participants ensured I knew them well and could study their interactions in depth. Other than these caveats, this research can be considered directly meaningful to a wide variety of organizations that produce software as well as designers and developers from a diversity of backgrounds.

Sufficient data for the type of research conducted were collected from 21 individuals in eight organizations. Overall, 150 hours of observations were made, 26 interviews were conducted with 21 people, and 13 meetings were video-recorded.

A variety of data sources (videos, photos, artefacts, videos and observation notes) were sampled to ensure the richness, breadth, and depth of the data; and also to achieve triangulation aims. To collect my data I used standard field study techniques (i.e., observations, interviews, and video recordings) but, in addition, particular attention was taken to collect information about artefacts. To this end, hundreds of material artefacts were photographed in the workplace or during artefact-mediated interviews and many more audible artefacts were captured in transcripts and recordings. The photos and videos were particularly helpful because of the pace and complexity of software creation work. Photos and videos allowed me to examine the activity afterward at a greatly reduced pace and ensured I understood the meaning of interactions and the usefulness of artefacts. I interviewed and photographed at eight organizations and observed and videoed at four of these. This produced ample data to analyze.

Truth value is another important criterion of qualitative research (i.e., ensuring the truthfulness of participants). Interviewing designers, developers and managers separately on the topic of a particular project ensured truthfulness, because I could readily identify any major discrepancies between accounts, and question its meaning. Further, particularly with respect to the four organizations where I also observed, the truth value of the interviews, in particular, was ensured by postponing the interviews until an extended period of observation with the participant had been completed, rapport had been established, and much knowledge of the projects obtained. I adopted this practice during the data collection period, as most of my participants were strangers to me prior to this

research. A further practice I adopted in each organization was to follow the designer first and then the developer. This allowed me to obtain a birds-eye-view of the project work before plunging into the details that developers address in their day-to-day lives. In this way, I was better able to understand the broader context of the developers' actions and therefore was able to interpret his or her words and actions more truthfully.

The quality of truthfulness can also be applied to other data sources besides interview transcripts. With regard to the photos, in order to be able to remember how each of the artefacts was used, all photos were annotated on the day they were collected. Also, field notes, which purposefully captured thousands of uses of artefacts (visual and aural), were enhanced with the photos and their captions, a process that helped me to immerse myself in my data and to increase the truth value of my notes prior to analysis.

To ensure a meaningful analysis, disconfirming evidence was sought. As I analyzed the data I tried to disconfirm theories I was developing mainly by sampling widely and organizing my data and the interpretations of my data in such a way that I could obtain a birds-eye-view of the results. This entailed creating many large intermediary analysis artefacts in the form of tables. Because I had so much data I needed to be reminded of other cases as I immersed myself in new ones and the intermediary tables helped me to do this.

Further, discrepant cases were sought and analyzed. For example, the developer whose Interactional Identity was "historical anomaly" was a challenge to categorize as either designer or developer because this individual worked right at the boundaries of design work and development work. However, because he mainly identified as a developer, I viewed him as one. At work this individual also struggled with defining himself within the team. His identity shows that he understood himself as being different

historically (i.e., his role was unlike any other roles he was aware of) and that this played a large factor in the way he interacted with others and used artefacts. Understanding how this individual addressed and understood his particular role on the team helped reveal biases in my understanding of what it is to be a designer or a developer. In addition, designers who worked mostly on content (e.g., the teacher/learner), and less on elements that required programming by a developer, were also challenging, and helped me to break down my preconceptions of what design work on a software project entails.

Finally, to further ensure meaningfulness, I thoroughly explored the context of designer-developer interactions as suggested by Morrow (2005), including their historical context, the meetings in which they interacted, the activity in which they were engaged, and surrounding activities that had an impact on the software creation work. A thorough exploration of context served to reflect the reality of software creation work from my participants' perspectives. The approach taken here to emphasize or highlight context, was similar to the approach taken in van der Riet's (2009) activity-theory based research. In this work, "context" was not conceptualized as a container for interactions, but as a mediating factor in both activity and interactions, and interactees and their social worlds were understood to be intertwined.

Adequacy of the Analysis

Morrow (2005) also identifies a number of criteria to assess the adequacy of the interpretation of qualitative data. For this research, the research analysis aimed to ensure ontological authenticity (i.e., that participants' individual constructions or views of reality were improved, matured, expanded, and elaborated). This was achieved by becoming deeply immersed in all of the data and by using qualitative methods conscientiously and carefully to ensure that participants' individual constructions were understood in depth.

In particular, the process of transcribing, entering, coding, organizing and reorganizing the data in ATLAS.ti ensured immersion in the data, as did the practice of reading and rereading transcripts and field notes. Ontological authenticity, understood in the broader sense in that the participants themselves developed their viewpoints, was not an aim of the research, but may yet be achieved to some extent when I report back to my participants, their organizations, and the research community.

Other forms of authenticity outlined by Guba and Lincoln (1989) and reaffirmed by Wardekker (2000), but from an activity theoretical perspective, that, like the broader understanding of ontological authenticity, are also more directed toward changes in participant's understandings or their social context, such as educative authenticity, catalytic authenticity, and tactical authenticity were also considered to be outside of the scope of the research aims.

Throughout, cultural-historical psychology was used to guide the exploration of the data and to shape the use of qualitative methods. In this way, the ontological and epistemological assumptions that are present in any research were exposed, and psychological understandings, garnered through decades of cultural-historical research, were leveraged to enhance the depth to the analysis.

The data were interpreted systematically through a well-planned research strategy comprising multiple studies. A major challenge of the data analysis was to identify techniques that could work for large amounts of data and the diverse types of data that were collected and then integrated together to create rich pictures of designers and developers interacting in each organization. To begin, the research questions were used to filter out all but the essential data. Then the data were analyzed using a variety of qualitative methods. Two methods were used directly as described in the literature: the

case study method and activity systems analysis. Unfortunately, because cultural-historical psychology, and more specifically activity theory, does not have a well-developed set of analytic methods, a lengthy process was required to review, select, and then adapt tools for some of the data sets. To this end, interaction analysis, grounded theory, and contradiction analysis were adapted and then applied. All of these adaptations were clearly described in this dissertation.

Whether used directly or adapted, all methods were used rigorously. When analyzing the interview transcripts in particular, written memos were created to help record and support the analytic process. Also, for almost every analysis, I created electronic spreadsheets and filled these with notes as a way of coping with the large amounts of data I had to process. I also used journaling during non-routine stages of research work.

When reporting the results I attempted to achieve a balance between investigator interpretations and the inclusion of data that would bring readers closer to the experiences of my participants. To bring the experiences of my participants directly to readers I used quotes, images, and descriptions of their interactions.

Consequential Validity

The consequential validity of the research has been demonstrated through the acceptance of papers that have been presented at conferences and published in proceedings and books on activity theory and on the software creation activity. Hence this foundational work on activity theory and the initial findings of this research appear to be interesting to the broader research community, designers, and developers. Conferences at which papers were presented include the Annual Convention of the Canadian Psychological Association (CPA) 2005, the International Society for Cultural and

Activity Research (ISCAR) in 2008 and 2011, and the International Society for Theoretical Psychology (ISTP) in 2005 and 2007. Papers appeared in the *Proceedings for the 5th World Congress on Management of Intellectual Capital and Innovation 2003*, and in the *Proceedings for Agile 2008*. A paper was also written for *The History and Philosophy of Psychology Bulletin* in 2005 and a chapter was written for the 2007 book *Citizen city: Between constructing agent and constructed agency*. I was also a keynote speaker for an international workshop on Design, Specification and Verification of Interactive Systems (DSV-IS) in 2008. These papers (and others I wrote previously from a computer scientist's perspective) all appear under the reference list under "Brown."

Validity and Reliability Revisited

Guba and Lincoln (2000) point out that the boundaries between paradigms are shifting and merging, and criteria for the evaluation of research is constantly evolving. It is therefore possible to also evaluate this research from a post-positivist paradigm to provide a different, and equally valid, perspective and to link this research to more classical research in psychology. To this end, Silverman (2005), addressing the issue of criteria for qualitative research, claims that research can be shown to have trustworthy conclusions and models if it can be shown to be valid and reliable.

This research is valid, in the sense that it effectively captures a particular view of reality (Silverman, 2005), because of a) its reliance on the method of constant comparison (case studies were tested against other studies of similar project work or meetings), b) the comprehensive treatment of the data (all cases were thoroughly analyzed), c) the incorporation of "deviant" cases into the analyses and therefore the conclusions and models to ensure generalizability of the results, and d) the appropriate

use of tabulations to convey properties of the data and the extent to which artefacts are used in designer-developer interactions.

Reliability (i.e., consistency and professionalism in the treatment of the data) is indicated by a) the high data quality that resulted from extensive sampling at multiple organizations and the use of professional transcription services for the interview and video transcripts; b) the significant amounts of original data included in the write-up to provide access to original data from all participants and projects; and c) the fact that all data were readily available throughout the entire analysis process because a single qualitative analysis tool was used to code and access all types of data, which allowed me to use consistent codes across all the data types and to test emerging hypotheses across all data types.

To complete the evaluation of this research, I now turn to Elliott, Fischer and Rennie's (1999) suggestion that research should also be evaluated in light of the research questions that motivated the research. This final way of evaluating the research is considered next through a critical summary of the work conducted in each of the four perspectives.

The Four Perspectives and Their Research Questions: An Assessment of Findings

The aim of this research was to understand the interactions of designers and developers engaged in the early stages of software-creation work and the role of artefacts in those interactions. This four-part study was designed around four research questions and provided four perspectives on the topic. The following remarks address each perspective in turn.

Historical Perspective

The first perspective addressed the question: What can be understood about designer-developer interactions from a historical perspective? To understand the role of the designer and developer I studied the disciplines that determined those roles. Then I studied the writings of researchers and some organizations that had worked on bridging the gap between these disciplines by developing new processes, joint artefacts, or by re-conceptualizing the work so that joint work could proceed smoothly.

By reviewing the literature on the disciplines of the designer and developer, I built up a picture of how these disciplines had developed over time. The analysis revealed a number of disruptions within the disciplines, including the rapid pace of change within both disciplines; constantly shifting boundaries (e.g. the context of use for software); rapidly evolving tools, including theories; the disputable nature of the design and development terrains; and the difficulty arriving at a unified process for creating software. Further, the analysis showed that communication through artefacts to resolve these issues is essential, but problematic, for both the designer and the developer. These disruptions revealed contradictions in the disciplines (listed in Table 9 and in Table 10) that carried over to contradictions in the roles of designers and developers.

Contradictions in the developer's role identified areas where the developer's background would be limiting in terms of assisting him or her to address the problems of developing software objects. The review stressed the challenges inherent in software work, the necessity of using artefacts in the work to make software visible, as well as the essential inadequacies of any artefact used for this purpose.

Contradictions in the designer's role addressed the nature of the designer's work including that much of it would of necessity be redesign work, that a connection with the

field could be challenging because the discipline is amorphous, that the application of theories could be challenging, and that design work required a significant amount of adaptation in the workplace.

Interdisciplinary designer-developer contradictions that were identified addressed the nature of the linkage between the disciplines showing that the work of designers and developers was inexorably intertwined, that terrains were disputable, practices constantly shifting, and processes ill-defined. In addressing these challenges and others, the analysis showed that designers and developers of necessity rely heavily upon artefacts, but the amorphous nature of software and the experiential nature of the interface make this communication essentially challenging.

The research contributed a number of unique and interesting insights on how disciplinary contradictions translated to role contradictions (see Table 9 and Table 10) and a list of interdisciplinary contradictions (see p. 131). Using a historical approach also resulted in a method for finding role contradictions that could be useful in other research.

Despite its utility, the particular historical analysis conducted for this dissertation had limitations. The analysis of disciplines is founded on a literature review that benefits from sociological and historical research, but because the disciplines I studied are new, there is little research of this type. Also, because of the breadth of the material in the area of design, some areas were inevitably de-emphasized. However, overall, it is worth noting that the contradictions identified in the roles and interactions were confirmed by the observations and interviews conducted in the research for Perspectives 2 and 3 and that this is reported in Appendix I and Appendix J.

Cultural Perspective

The research conducted from a cultural perspective aimed to answer the question: How is the work of designers and developers structured within organizations and what role do artefacts play here? Workplace observations in four organizations and an analysis of designers and developers at work exposed the activities in which the designer and developer engaged. This provided a clear answer for their reasons for the artefact-mediated interactions—they contributed to achieving the outcomes and objectives of these activities.

A case study of Talia showed that the designer and developer were involved in two activities: creating software and managing the software creation activity. Examining these activities in detail exposed artefacts that were essential to the work, such as drawings, demonstrations of software products, abstract models, and exemplars. The analysis identified all the elements of an activity and these were discussed in relation to the activity's aims. These aims were revealed through five interviews, four with members of the Talia software-creation team and one with a Talia analyst. Explicating these aims was essential for understanding what the artefact-mediated interactions were meant to achieve. Although all aims were not equally important and therefore received different treatment, Table 12 showed some strategies for achieving aims, such as the creation of specialized roles on the team, and the use of bridging artefacts and bridging individuals.

A limitation of this part of the study was that some interesting avenues could not be explored fully, such as the role of business representatives (e.g., product owners) who had a strong influence on the entire activity, including designer-developer interactions.

The research in part 2 of the cultural perspective examined how activity systems outside of the creating activity influenced the social and cultural objectives (usability,

maintainability, etc.) that would be incorporated into the software, thereby exposing how software comes to have certain qualities. Focus-On charts were developed for all four of the observed software-creation activities. These revealed the influence of other activities (past, present, ongoing, and anticipated) on the actions of the designer and the developer. In particular, this analysis exposed how certain roles and artefacts facilitated the coordination of different activity systems surrounding the software-creation activity. This showed where certain software aims originated and the resources employed in their achievement. These resources were often mediators in designer-developer interactions.

The research in this second perspective contributed a solid analysis of the activities in which designers and developers jointly engage, which provided the context and the reasons for the artefact-mediated interactions. Focus-On charts were developed and these depicted the organizational mechanisms for ensuring the achievement of important social and cultural objectives of the software that was being created through the software-creation activity.

The limitations of the research in Perspective 2 were related to the limitations of the data used for the analysis. In all cases, my understandings of the work structure were developed in less than nine days spent on a site, observing only concept design work, and through interviewing only the designers, developers and managers. While substantial, these data do not reflect the entire spectrum of software creation work. The information in the Focus-On charts therefore reflects these biases. Also, much of the information in the Focus-On charts comes from second-hand knowledge about surrounding activities because I did not observe these directly. The charts therefore primarily reflect “perceived influences” from the perspective of the designers and the developers.

Although not without its limitations, this cultural perspective provided a thorough understanding of software-creation activity in four organizations, through the application of activity systems analysis in one case and informal descriptions in the other three cases. This, in turn, provided insight into differences and similarities. In particular, all sites relied heavily on artefacts to accomplish software creation work, although each site used completely different technologies and relied upon different preferred types of artefacts. The study linked the use of artefacts with the achievement of aims in an activity, and showed many links between artefacts and aims. In addition, the Focus-On charts are a novel contribution to research in this area. These charts summarized a considerable amount of information and facilitate reflections on how the perceived or real aims of activities outside of the software creation activity influence the software creation activity itself.

Participant-Centric and Situated Perspective

The question addressed in this third perspective was: What meaning do artefact-mediated interactions have for designers and developers? Twenty-six in-depth interviews with twenty-one individuals and an analysis of twenty of the corresponding interview transcripts revealed the personalized and situated meaning of artefacts for designers and developers. Using an adapted version of grounded theory to analyze these data, I made a case for an abstract concept called Interactional Identity, which I specified for each participant. This enacted identity was evident in an individual's team interactions. In most cases, these identities were confirmed through observations.

The Interactional Identity itself was revealed indirectly through the participant's interviews and expressed clearly through four coded categories that captured comments participant's made about their personal goals for the project, their opinion of who owns

the user interface, their views of design artefacts, and their views of project contradictions/tensions. Of particular interest to this dissertation was the fact that Interactional Identity explained how specific designers and developers used artefacts to advance work on the user interface in order to incorporate goals they saw as important. For example, for Owen the movie director, artefacts were treated much like props in the larger drama of the team's game design work that would create the game. For Wil, whose Interactional Identity was binder, artefacts created by other team members were treated as specifications to be amalgamated and integrated into the software, a process that ensured the application would slowly come into existence.

Just as in the case of Owen and Wil, each of the other 18 designers and developers had a unique Interactional Identity that was shaped by their discipline, their past, and their current circumstances (see Appendix F for a description of these identities). Like Owen and Wil, their Interactional Identity provided an overarching approach to the use of artefacts in software creation work.

I also explored the nature of the concept Interactional Identity by reflecting on the data from all 20 participants. My results here suggest that Interactional Identities develop slowly and that the Interactional Identities of individuals with less than 10 years of experience were significantly different from those of individuals with more than 10 years of experience. However, a clear demarcation between designers and developers with and without 10 years of experience seems unlikely, and the ten-year boundary is interpreted here more as a point in time around which such a transition occurs, with some variability accepted as inevitable. As shown in Figure 11 and Figure 12, more experienced designers and developers expressed through their Interactional Identities the importance of collaboration within their teams. Further, their Interactional Identities provided detailed

explanations of how they worked collaboratively. Results showed that each experienced designer or developer found a unique way to work collaboratively through the use of artefacts, which were essential elements of the work of every participant. Both Owen, an experienced designer, and Wil, an experienced developer, had Interactional Identities that assumed that software-creation work was collaborative.

The findings of this particular study were affected by the lack of diversity in the sample of participants. Effectively, any time both a designer and a developer agree to being interviewed and observed, and their managers (one to three individuals) also agree they may be interviewed and observed, the pair is likely to have a good working relationship. The Interactional Identities may therefore be biased toward functional, as opposed to dysfunctional, Interactional Identities and may not represent the entire spectrum of possible Interactional Identities.

This part of the dissertation research led to the detailed explication of a new theoretical concept that encompasses role, action, and personality. Moreover, the actual identities associated with this concept provided insight into designer-developer interactions, while the exploration of the relationships between Interactional Identities and other concepts, such as roles or projects provided a rich contextualized understanding of this concept.

The analysis of Interactional Identities developing over an extended period of time provided useful insight into both the work of creating software and how designers and developers come to understand their work as collaborative work over a period of approximately a decade. Knowledge of Interactional Identities provided insight into the meaning that artefacts have for designers and developers.

Artefact-Centric Perspective

Perspective 4 examined how artefacts supported designer-developer interactions in meetings. The primary means of answering this question was through an analysis of videos and photos of the artefacts that the designers and developers used when working together in meetings. However, in addition, the research notes were invaluable for providing the context in which artefacts were used and the interviews helped reveal the value of artefacts to designers and developers.

The first study using this artefact-centric approach, I used grounded theory to analyze artefacts represented by photos and images embedded in their context of use as captured in my research notes, interview transcripts, and video transcripts. The result was a purpose-based classification scheme consisting of a dozen artefact classes used to mediate designer-developer interactions during software-creation team meetings. Although the specific artefacts and kinds of artefacts used varied from one organization to the next, this study showed there is a great deal of commonality in the purposes of artefacts that are used in concept design meetings. A great deal of commonality therefore also existed between the organizations when examined from this perspective.

My coding evolved through several stages and the final version could be further refined. For instance, the category “stories” seems too tied to the form of the artefact (i.e., a narrative form), rather than to the purpose. While I did identify stories in different forms (oral, written, and tabular), I believe there is much to be explored in terms of the varied purposes of stories. I also found that the classification scheme missed highlighting the benefits of unique artefacts that failed to form a category, such as the tickets used at Telespace.

In the second artefact-centric study, I analyzed a series of meetings at Jaba using interaction analysis. Each of these meetings involved both a designer and a developer. The results showed that meetings were typically initiated because of a product or project tension and meeting attendees expressed their concerns through their talk and through artefacts. Meetings frequently ended with team members leaving with a deepened understanding of many issues and resolutions to some contradictions.

In the first part of the third study, I thoroughly analyzed a two-hour video taken in one organization. This analysis revealed that meetings involving designers and developers were structured in phases. Phases were determined by breaking up the meeting at points when the team's attention moved from one set of artefacts to another set. Phases were named by identifying the goal of each artefact-mediated interaction in a phase, and naming the phase according to the most common goal. An analysis at the operation-level revealed a link between the form of an artefact (e.g., list, story, sketch) and its ability to support cognitive functions (remembering, indicating, judging, and so on).

In the second part of this study, I focused solely on meeting phases and the artefacts used in those phases. I analyzed approximately 11 hours of video data, comprising 13 meetings from four organizations. From these data, a model of concept design meetings involving designers and developers was developed (see Figure 43). Phases were repeated in meetings and a final set of commonly occurring phases was identified. However, the presence of several unusual phases suggests that the set of phases identified here may not be definitive. Of particular interest was the presence of project assessment phases and smaller evaluating subphases found in most phases, both of which occurred frequently in all the meetings I observed. These showed how designers

and developers addressed the aims of their activity. Large assessment phases occurred periodically in almost all meetings. At these times, the team addressed the needs of the team, the client, the business, and end users collectively, making tradeoffs if necessary. Smaller evaluation subphases addressed the issue of incorporating or balancing the multiple aims for the software.

Linking the results from the three artefact-centric studies resulted in the expansion of the model to show the typical artefacts that were used in each phase (see Figure 44). The combined model suggests a number of phases that are beneficial to software-creation work, the nature of the activity, and the artefacts that support it. These are phases I regularly observed when designers and developers interacted in meetings. Common meeting phases were Simulating, Presenting, Enacting, and Exploring, each of which was supported by a distinct set of artefacts.

Overall then, this approach indicated that artefacts helped focus and execute the required work during the various phases of designer-developer meetings.

This artefact-centric approach, however, was limited in part by the types of software projects that were being undertaken. Many types of products were not included, such as the development of control systems, applications that must achieve very high standards of quality, real-time applications, very large systems, or applications in specialized domains (e.g., mobile applications). Similar studies of different types of projects may well challenge or extend the models developed here; but the models are a significant contribution, because they were developed from empirical data, and a good reflection of the nature of the concept design work that designers and developers undertake. These models were developed from data from four diverse organizations and

are likely relevant for a significant range of real, novel, and challenging software projects that would take several months to a year or more to develop.

Overall, this dissertation has shed light on the conflicts, influences, and challenges that designers and developers, as part of multidisciplinary software creation teams, face and overcome. The four parts of this research explored the role of artefacts in this work from four very different, but complementary perspectives: (a) a historical perspective, (b) a cultural perspective, (c) a personal and also situated perspective, and (d) an artefact-centric perspective.

The historical research emphasized the challenges presented by the amorphous nature of the software object, the continuous evolution of that object over decades, and the contradictions inherent in the roles of the designer and developer that have evolved to work within this challenging domain. Adopting a cultural perspective revealed that software-creation work is really a network of activity systems and the importance of making strong connections between activity systems to achieve important aims. The personal and situated research showed that designers and developers, some of whom are managers, enact Interactional Identities that determine their approach to the use of artefacts within the team. It also showed that more experienced designers and developers enacted Interactional Identities consistent with the collaborative nature of their work. Finally, the artefact-centric research revealed that common kinds of artefacts are used in designer-developer interactions. The most commonly used were interface proxies, stories and meetings (the latter being understood as artefacts or tools used by designers and developers). These reflected the vital importance to designers and developers of simulating software interactions in meetings that advance software-creation work. However, phases other than simulating were also important, as were other artefacts.

Implications of the Findings

The previous section reviewed and critically evaluated each perspective. This section looks at the results as a whole and outlines the implications for theory, methods and practice.

Integration of the Results

The historical review revealed ambiguity in the specification of the designers and developers' role, and justifiable reasons for this. It also showed that there is considerable ambiguity with respect to exactly how designers and developers should interact and the types of artefacts that may facilitate their interactions. It is therefore possible that designers and developers construct and enact their Interactional Identities in response to these ambiguities. In effect, an Interactional Identity may be a designer or developer's solution to how to design or develop within a team because it provides an overarching framework or narrative for effective artefact-mediated, inter-team member interactions.

The cultural analysis showed that software projects have multiple, potentially competing aims that provide a framework that drives and constrains the team's creative work. Artefacts were shown to have a role with respect to exposing solutions to the achievement of an aim, aiding in the revelation of aims, and helping to resolve conflicting aims. However, the analysis showed there were many ways aims could be lost, misunderstood, misinterpreted, inadvertently overlooked, inappropriately prioritized, difficult to measure, or unrepresented. These results suggest that Interactional Identities may also provide designers and developers with a comprehensive strategy for addressing the challenges of dealing with the multiple and conflicting software aims that are a part of the social milieu of project work. An Interactional Identity may provide a framework for

understanding and dealing with software aims in general and may allow an individual to filter and prioritize the many project aims in a way that makes sense given their role.

Considering the results of the cultural and participant-centric studies side-by-side exposes the dependence of the designer and developer's achievements on their social milieu. Specifically, the results suggest that an environment where there is adequate structure and where software aims are clearly exposed, resourced, and tracked is one where the designer and developer are more likely to create effective and valued software. Designers and developers appear to achieve this both through their individual actions and through the team member interactions that are mediated by their Interactional Identities. This points to the importance of social structures and the role of other team members in the successful creation of software and exposes the symbiotic relationship that exists between the designer and developer, whose actions directly result in the creation of the software, and the rest of the team.

The cultural and artefact-centric analyses also showed that despite much dissimilarity between types of designers, types of developers, software projects, organization types, and software processes, there were many commonalities between all designer-developer interactions and the activities that these interactions were directed toward. For instance, the cultural analysis showed that all interactions occurred within a network of activity systems that, in various ways, entered into the interactions of the designer and developer. Evidence of this is found in the designer and developer's conscious awareness of these multiple "surrounding" activities and the ways these activities influence and impact the aims of the software they work to create. The artefact-centric analysis exposed another commonality, namely that meetings of various types were always a primary tool (i.e., artefact) for joint, focused work and that they comprised

a number of commonly occurring phases (i.e., presenting, simulating, enacting, exploring, and addressing the project aims as a whole). Further, although not depicted in the model of meetings, the analysis revealed that all of these phases were regularly interspersed with “evaluating” subphases. In these subphases, the project team members, but especially the designer and developer, resolved and addressed conflicting aims, many of which were highlighted in the cultural analysis. However, the research also uncovered a large discrepancy between the total number of aims to be addressed in any given software project and designers’ and developers’ ability to address multiple aims at any given point in time. For example, the analysis of Jaba meetings showed that typically two or three aims were addressed in any given evaluation subphase and that a total of 12 different aims were raised over the course of a two-hour meeting, a total which comprised many, but not all, of the project aims revealed in the Jaba analysis conducted for the cultural perspective. Meetings therefore regularly addressed only a subset of the aims and only in certain restricted combinations. The reality for designers and developers is that addressing and balancing the aims of the software is a challenging, constant, and pervasive task of software concept design work, as evidenced by the fact that evaluation subphases are embedded in all phases of the work.

Theoretical Contributions

This dissertation research led to several theoretical contributions. The grounded theory analysis of the participant-centric perspective, led to the development of the concept of “Interactional Identity,” which overlaps with a number of other well-developed concepts in activity theory such as personality (Leontiev, 1978c), role (Engeström, 2000) and action (Wertsch, 1991). This concept has links to Bucholtz’s (2005) work, which she conducted from a socio-cultural perspective. Bucholtz theorizes

identities as “the social positioning of self and other” (Bucholtz, 2005, p. 586). She says identities are a relational and socio-cultural phenomenon that emerge and circulate in local discourse contexts of interaction, rather than a stable structure located primarily in the individual psyche or in fixed social categories. This dissertation supports these claims, provides substantial empirical evidence, and contributes significant additional insights about Interactional Identities, and in particular their tie with artefacts (i.e., that Interactional Identities structure an individual’s use of artefacts). This dissertation also shows how such identities develop over time in adults engaged in a creative occupation that requires significant amounts of education. This aspect of the results may also be related to Schön’s (1983) two step model of the “reflective” practitioner, particularly in its description of the Model 2 practitioner who is much more collaborative in their approach to work in comparison to the less-experienced Model 1 practitioner who operates with a win/lose, i.e., right/wrong, paradigm. My research, which provides empirical evidence of the identities of practitioners in action, might therefore be of interest to developmental or organizational psychologists.

Interactional Identity was a concept elaborated upon in this dissertation because of its utility for developing a complete understanding of the use of artefacts in the workplace. An Interactional Identity is an outcome of the software creation process and is enacted in team interactions. However, over time individuals enact more collaborative identities, which appears to come about in part through an individual’s struggle with their ego (see Table 20), suggesting that aspects of an Interactional Identity are a part of the self.

This dissertation speculated that the reasons for developing an Interactional Identity were that it filled a gap between a prescribed role and practice, and also that it

provides a strategy for an individual to contribute to the artefact-mediated work of the team, whose aim is primarily to fulfill the cultural aims for the software (usability, learnability, and so on) and the project work (completes on time, satisfies the client, and so on).

In line with other sociocultural research theorizing about identity, identities are seen in this dissertation as constructed through interactions with others. Like Holland and Lachicotte's work (2007), identities are seen as "ways of inhabiting roles, positions, and cultural imaginaries" (p. 103) that are multifaceted and changing (Lemke, 2003). As in Swidler's (1986) view, "strategies of action", i.e., Interactional Identities, are developed from cultural components and explain individual action.

However, there are some significant differences between Interactional Identities discussed in this research and identities, such as "villager", "educated person", "military person", "Dane", "logger", or "environmentalist" discussed in the socio-cultural literature (Holland & Lachicotte, 2007). Those identities clearly apply to, and are constructed by, groups of people. Interactional identities are more tied to the individual and express an individual's agency in the context of an activity in which that individual is engaged. The treatment of identity in this dissertation also stresses the relationship with artefact use, and copious evidence is provided for identities imbuing artefacts with additional meaning. Also this research provides a detailed understanding of the components of an Interactional Identity and how it is and is not expressed. It is also clear on research methods for revealing such an identity, suggesting the use of interviews backed up with observations. Further, it suggests some questions that might be asked to discover an Interactional Identity such as: What are the individual's personal goals in the context of this specific activity?, What are their views on who owns the object of this activity?, How

are artefacts useful to them when communicating to others engaged in the same activity?, and What are the roles of contradictions/tensions in an activity?

Another theoretical contribution was the development of the concept of “phase” from the artefact-centric analyses. Phases were conceptualized as a sequence of actions or interactions that are focused around a single set of related artefacts and generally oriented toward a dominant goal or joint goal. The notion of a phase has been used informally by many activity-theorists, but has never been well defined. Leontiev (1977) used the loosely-defined concept of “moment” to refer to a series of “subject-activity-object transitions” that are imbued with drama and are meaningful for the subject. In activities that are rich with artefacts, identifying phases of an activity (by identifying sets of related artefacts around which the activity is centered) may be a useful approach for understanding an activity. This dissertation moves toward developing an understanding of the concept of “phase” by providing a possible definition, many examples, and by using phases in an integrated model of software creation activity. In this dissertation, a consistent understanding of phase was useful for revealing the various aspects of a complex activity and was therefore a useful conceptual and descriptive aid.

Finally, this research makes a theoretical contribution by suggesting a way of classifying artefacts based on their purpose within a particular type of activity. This approach stands in comparison to Vygotsky’s (1999c) contribution of classifying artefacts generally as signs or tools (1978b), Wartofsky’s (1973) classification of artefacts as primary, secondary, and tertiary, and Engestrom’s categorization scheme of artefacts as what, how, why, when, and where artefacts. This work suggests there is also value in classification schemes that are associated with specific types of activity, and not just to activity in general. The classification scheme presented here builds on Cole’s

(1997) work where he presents an understanding of artefacts as both material and abstract, and where he proposes aural as well as visual artefacts. It also has links with Newman and Holzman's (1993) conceptualizations. They describe a category of tools called tools-for results, which are artefacts that are both tools and objects of activities. As in my research, many of Newman and Holzman's artefacts are cognitive tools such as concepts and ideas. However, because of the nature of the software work, this dissertation also identified some artefacts that were purposefully shaped to support software creation work and were dynamic or "experiential." For example, interactive prototypes, a common artefact, can only be experienced over time and through active engagement of the perceiver. This led to the various types of design meetings also being classified as artefacts because they are experiential, have abstract and material elements, and are a common tool in design work. The idea of "experiential" artefacts was not fully explored, but is provocative. The idea of meetings as artefacts, though perhaps not intuitive, and possibly problematic, is at least intriguing.

Incorporating all of the previous concepts, this dissertation also makes a contribution toward understanding the consequences of artefact-use for cognition. Vygotsky posited that the mediational nature of artefact led to qualitative shifts in cognition relative to cognition without artefact use i.e., more can be accomplished cognitively-speaking, with the aid of meditationally- appropriate artefacts than without them (Bakhurst, 2007). This view was developed largely in the context of Vygotsky, Luria and Leontiev's research with children that showed that in the development of higher-level mental functions in individual children the mastery of cultural artefacts was a key element. The impression Vygotsky left was that adults were individuals with well-integrated, well-developed functional abilities who were generally experts in employing

culturally-appropriate artefacts to transform both their external world (through creative action) and their internal worlds (by extending their cognitive abilities) (John-Steiner, 2007). This dissertation has shown that artefact use is ubiquitous in the early phases of software creation work, arguing for their necessity and value in this creative and highly cognitive activity. The question then arises as to the purposes that these artefacts serve at the following three levels: operations, actions and activity.

At the operational level, artefact use, in combination with other means of communication, were shown to support interacting individuals in remembering, self-regulating, comparing, indicating, focusing, making associations, judging, expressing an idea, exploring, checking, and also keeping the activity interesting and engaging. In this sense, this dissertation documents the use of artefacts to enable higher-level mental functions in joint interactions.

At the level of human action, this dissertation showed that solving the problem of what software to create was a cognitive task that advanced through both individual and collaborative work, both of which were artefact-rich. The collaborative work was presented as work that was more disruptive of the activity; meetings, in particular, were shown to have the important role of revealing the desirable or necessary cultural attributes of the software and redirecting the work of individuals. Meetings proceeded in phases (sequences of group actions) at all four organizations. Despite apparent disorder within the meetings, common phases in the work that comprise the cognitive activity of software creation were determined to be simulating, presenting, enacting, exploring, and finally, assessing the project. Each of these phases was centered and organized around a set of predictable and simple artefacts that enabled their execution.

At the level of human activity, this dissertation showed that one outcome of software creation activity was the creation and deployment of an interactional identity, an identity enacted by designers and developers that helped to organize the work strategically (e.g., for the movie director their organizational strategy would be “when interacting with others I direct”, or for the binder it would be “when interacting with others I receive their artefacts and bind them”). Interactional identities were shown to reveal the designers and developers’ strategic use of artefacts in general i.e., the meaning they gave artefacts. Interactional Identities were shown to develop/transform over the period of approximately one decade and this, therefore, provides evidence of adult cognitive abilities developing over time. Specifically, the dissertation showed that a cognitive development of interacting adult developers and designers was the progression from task-focused thinking to collaborative thinking. This corresponding change in thinking also gave artefacts in the workplace a different meaning. In experienced designers, artefacts were understood, in part, in terms of their collaborative potential.

Cognitively, therefore artefacts were seen to support operations, actions and activities, and at each level they provided a qualitatively different purpose.

Research Methods

An ethnographic approach was taken to studying this topic as suggested by Jordan (1996), who says this approach is well suited to developing a better understanding of complex work, and that participant observation, in-situ question-asking, and micro-analytic methods of analysis all help to deepen such an understanding. Research in complex domains, she says, “must be grounded in a thorough understanding of ongoing work practices and how they are supported (or not) by the physical layout, artifacts, information systems and data bases, as well as the social relationships and arrangements

of the workplace” (Jordan, 1996, p. 17). This approach, taken in this research for data collection, was supplanted with analytic methods aligned with the paradigm of cultural-historical psychology and, in particular, activity theory.

This research suggests that qualitative methods are a valuable tool for activity theorists, as previously demonstrated by Roth’s (2005) studies of scientific activity and educational studies of other researchers recently reviewed (For a review see Wilson et al., 2010). Concurring with Jonathan Smith (J. A. Smith, 1995), Wilson et al. have suggested that qualitative methods are particularly helpful for addressing issues related to exploring, describing, and interpreting the personal and social experiences of participants, as has been demonstrated by this research. They also explain how using activity theory with qualitative methods does not mean that qualitative research is conducted from a preconceived non-activity theoretical stance. The alignment of methods and perspective was achieved here through the modification of qualitative methods to align them with cultural-historical psychology.

This research identified a need for qualitative methods that were aligned with a cultural-historical, activity-theoretic paradigm and that were of immediate practical use for activity theorists. Because of a paucity of such methods, after a careful process of selection, suitable qualitative methods with similar, but different, theoretical underpinnings were adapted to align with cultural-historical activity theory. Such adaptation of research methods is common in qualitative research as shown, for example, by the many variations of grounded theory that have been spawned when different paradigms or philosophies of science were adopted by researchers using this analytical technique (Ponterotto, 2005).

In this research, contradiction analysis was developed from Turner and Turner's (2001) version to expose contradictions in roles and in interactions through a historical review. Grounded theory was adapted to fit with activity theory, rather than Strauss's (1993) theory of action. Interaction analysis was modified to align with activity theory as well as with the social circumstances surrounding dissertation work, particularly its solitary nature.

In conducting this research the focus was on interactions and activity. Typically, in activity theoretical research, the focus is on the individual engaged in a specific group activity, although research focused on interacting individuals can also be found. For example, Engeström, Brown, Christopher, and Gregory's (1997) work in the courts created a categorization of types of interaction (coordination, cooperation and communication). Wells (2002) developed a model of interaction by studying two girls working on a school project. This current research provided insight on various ways to study the general topic of interaction, while also emphasizing the role of artefacts as mediators of interacting individuals. For example, the historical review indicated a strong influence of disciplinary activity on designers' and developers' interactions in a professional context. This suggests that for activity systems that include individuals playing a professional role, an historical analysis of the disciplines that shaped their role, as opposed to the more typical analysis of the previous forms of a specific activity, could be a fruitful avenue to explore. Such an analysis has the potential to reveal valuable information such as why certain types of artefacts are a part of the interactions of individuals from different disciplines, which is information that probably would not be revealed by analyzing previous forms of the specific activity being studied. The cultural analysis, with its emphasis on the social and cultural aims of the software object,

provided a good example of a way to link the study of the interactions of individuals to group activity, and to show their inter-relatedness. Further, the participant-centric perspective showed that, paradoxically, interactions can be studied from an individual perspective. This analysis revealed how differently each “interactee” may view their interactions and the overall strategy they use when incorporating artefacts in their interactions with others. The artefact-centric perspective showed that a focus on artefacts can reveal insight into the purpose of interactions, which in this case was achieved through a purpose-based classification of artefacts.

Practical Implications

Although not the aim of this research, the research findings have a number of potential practical implications. In the area of education, the findings suggest that it is important to teach designers and developers, from the beginning, how to work in multi-disciplinary teams. New designers, in particular, were shown to experience substantial frustration aligning their work experiences with their education. Evidence for this claim is found in the analysis from the participant-centric and situated perspective on Interactional Identities showing that frustration was a common emotion in designers with less than approximately 10 years of experience. This suggests a large gap between education and practice. The historical review showed that each discipline currently develops largely in isolation from others and the issues that relate to the integration of multidisciplinary work, which is currently learned during the course of a designer or developer’s career, are largely unaddressed in either undergraduate or graduate programs. This research also suggests that teaching designers and developers about cooperative work in teams and the social factors (as opposed to the technical concerns) that impact their work may also be of value. As such, tools like the Focus-On chart could be useful in

educational settings for reflecting on social and cultural issues that can impact the outcome of software products.

At the discipline level, the well-known gap between theory and practice has been articulated in terms of detailed and specific lists of unresolved contradictions both in the roles of the designers and developers and in designer-developer interactions. The deeper understanding of team interactions acquired through this research could lead to new tools and processes and could contribute to research already under way in this area. For example, the research suggests a need for tools that support team members to track and balance software aims and to understand these in the context of the social milieu in which the software is created, because this has a large impact on whether or not aims are attended to in concrete software creation practices. Also, the research suggests that software processes might fruitfully be expanded to address issues to do with integrating the work of all the team members.

Practically speaking, greater awareness of the importance of meetings, an important outcome of this research, could change the frequency of meetings, their constitution, the preparation of artefacts for meetings, the physical layout of real workspaces and meeting rooms, people's desire to reflect on meetings, or the way meetings are arranged and conducted in workplaces where designers and developers are unable to interact directly. Awareness of the common phases in meetings in which designers and developers engage could also help managers or teams reflect on and organize their daily activities in order to support the full range of potential reasons for meetings between designers and developers.

Deepened Understandings

The primary aim of this research was to deepen the understanding of designer-developer interactions within the software creation activity. As such this research stands beside empirical work conducted in other paradigms such as Suchman's (1987; 1995) Spinuzzi's (2003) and Robertson's (1997), which was discussed in the introduction.

However, this research also contributes to a body of research on software design and development, which although not empirical, was conducted from a cultural-historical perspective by researchers with familiarity or experience with either the design or development aspects of software creation work. Béguin (2003; 2007; 2009), for example, theorizes about end users and designers. Bertelsen (2000) has reflected on the phases of design work and about design artefacts as boundary artefacts that mediate the heterogeneity between designers and developers. Barthelmess and Anderson (2002; 2003) conducted work theorizing the co-constructive, collaborative aspects of software development, which they discussed using an activity-theoretical framework. Bodker (1991; 1998) worked on conceptualizing the user interface and the role of representations in design work and she and Bannon (1991) worked on theorizing the software object as understandable only in use. Chaiklin (2007) worked on conceptualizing design and practice (i.e., software in use) as integrated and intertwined systems. A variety of international researchers produced an edited collection of articles on the social aspects of software practice, which used activity theory as the primary paradigm (Dittrich, Floyd, & Klischewski, 2002). And finally, Kuutti (1996; 1999) suggested ways of conceptualizing the software creation activity.

This work is also similar to some empirical work conducted from an activity-theoretical perspective. Tuikka (2000; 2001; 2002) conducted research in a lab

environment on concept design work in multidisciplinary teams; however, except that the participants were real specialists, the rest of the work was not real. Warr and O'Neill (2006; 2007) also conducted their work in a lab environment with a design problem determined by the researcher, but from a "communities of practice" perspective; they conceptualized design work as creative and social, and they studied the phases of design work and artefact use. My research also has links with Miettinen and Hasu's (2002) empirical work on user needs articulation in product design work, which describes how product, networks of user activity systems, and user activities coevolve. Although they did not observed work directly, Miettinen and Hasu held seminars involving designers and end users and used activity theory to produce models, such as a model of the future work that would be required of the end users using a product under real development. The models focused on historical and cultural elements, and identified contradictions; the work exposed a weak link between the future end user activity with the new product and the design work of the team, which the researchers helped their participants to address through the development of insightful models.

In comparison to much other research on software creation work, my work differentiates itself in that its data were derived directly from real workplace environments and analyzed using qualitative methods. It also is unique in the number of projects that were studied simultaneously. These data and their analysis provided substantial evidence for claims made in the research, allowed for nuanced interpretations of artefacts, actions, and activities, and allowed generalizations to be made. This work also differentiated itself by emphasizing designer-developer interactions in software creation work.

Future Research and Closing Remarks

Future research could confirm theoretical findings in other situations or domains. In this regard, the concept of Interactional Identity could be further investigated within software creation activity and in other domains. Many ideas and question arise, such as longitudinal studies, studies of Interactional Identities of other team members, studies of Interactional Identities within dysfunctional teams, studies of Interactional Identities and culture in the broader sense of the word, developing simple methods for arriving at Interactional Identities, speculating about their value, and so on. Further, the models of meetings and the categorization scheme for artefacts could be tested and extended against software creation scenarios not considered in this research, such as the development of free software or software developed in situations where team members do not work face to face. In these contexts, one might ask if the phases/cognitive steps in the activity differ. Further, research could be conducted to assess the artefacts that are found in a workplace for the ability to support the work of software creation, potentially identifying weak or missing artefacts. Because a number of qualitative methods were adapted for the use of cultural-historical psychology, another line of research that could be developed is testing the adapted methods in other contexts and seeking other compatible methods to adopt to further extend the cultural-historical psychologist's toolset. Another line of research could consider the implications of the historical review, in particular, for the development of new curricula for designers and software developers. And finally, the detailed results on meetings and artefacts used in meetings could be used to draw attention to the importance of such meetings and further extend the tools and processes that occur within meetings.

This research makes the valuable contribution of a rigorously-conducted, theoretically-grounded study of designers and developers engaged in the early stages of software-creation activity directed toward creating real, novel, and useful software. The results collectively contribute to a much richer understanding of this domain and the designers and developers who address the diversity of social and cultural objectives that modern software demands. This research revealed a gap between disciplinary activity and practical activity. It exposed the social systems surrounding software development work. It detailed individual strategies, such as the creation of Interactional Identities that shape the interactions of designers and developers with each other, and with other team members. It documented and described the roles of artefacts in the joint work of designers and developers, work that regularly unfolds through meetings that support the exposure and resolution of product and project contradictions. This deepened understanding has potential for a number of practical applications.

As well as contributing a deepened understanding of software creation work, this research also made some theoretical and methodological contributions of potential interest and value to researchers in the field of cultural-historical psychology.

References

- Adler, P. A., & Adler, P. (1994). Observational techniques. In N. K. Denzin, & Y. S. Lincoln (Eds.), *Handbook of qualitative research* (pp. 377-392). Thousand Oaks, CA: Sage Publications.
- Alexander, C. (1979). *The timeless way of building*. New York: Oxford University Press.
- Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A pattern language: Towns, buildings, construction*. New York: Oxford University Press.
- Allen, F., Boehm, B., Brooks, F., Browne, J., Farber, D., Graham, S., Gray, J., Hawthorn, P., Kennedy, K., Leveson, N., Nagel, D., Neumann, P., Parnas, D., & Wulf, B. (1999). *ACM panel on professional licensing in software engineering: Report to council*. Unpublished manuscript.
- Ambler, S. W. (2007). *Generalizing specialists: Improving your IT career skills*. Unpublished manuscript. Retrieved June 25, 2010, from <http://www.ambysoft.com/books/agileModeling.html>
- Ambler, S. W., & Jeffries, R. (2002). *Agile modeling: Effective practices for extreme programming and the unified process*. New York: Wiley Computer Publishing, John Wiley and Sons, Inc.
- Anderson, J., Fleek, F., Garrity, K., & Drake, F. (2001). Integrating usability techniques into software development. *IEEE Software*, January/February, 46-53.
- Bakhurst, D. (2007). Vygotsky's demons. In H. Daniels, M. Cole & J. V. Wertsch (Eds.), *The Cambridge companion to Vygotsky* (pp. 50-76). New York: Cambridge University Press.
- Banister, P. (1994). Observation. In P. Banister, E. Burman, I. Parker, M. Taylor & C. Tindall (Eds.), *Qualitative methods in psychology: A research guide* (pp. 17-33). Buckingham, UK: Open University Press.
- Bannon, L. J., & Bødker, S. (1991). Beyond the interface: Encountering artifacts in use. In J. Carroll (Ed.), *Designing interaction: Psychology at the human-computer interface* (pp. 227-253). New York: Cambridge University Press.
- Barbacci, M. R., Chang, C., Coallier, F., Howell, C., Kark, A., Kruchten, P., LeBars, L., McConnell, S., Nash, D., Otto, F. & Reeker, L. (2005). *Guide to the software engineering body of knowledge*. Retrieved February 18, 2006, from <http://www.swebok.org/>
- Barthelmeß, P. (2003). Collaboration and coordination in process-centered software development environments: A literature review. *Information and Software*

- Technology*, 45(911), 928.
- Barthelmess, P., & Anderson, K. M. (2002). A view of software development environments based on activity theory. *Computer Supported Cooperative Work*, 11, 13-37.
- Battarbee, K. (2004). Co-experience: Understanding user experiences in social interaction. (PhD, University of Art and Design). Retrieved from <https://www.taik.fi/kirjakauppa/images/2be572c773f32c5b5450d0b313a02c65.pdf>
- Beck, K., & Andres, C. (2004). *Extreme programming explained: Embrace change* (2nd ed.). Reading, MA: Addison-Wesley Professional.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cuningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). *Manifesto for agile software development*. Retrieved January 23, 2006, from <http://www.agilemanifesto.org/>
- Becker, H. S., & Geer, B. (1960). The analysis of qualitative field data. In R. N. Adams, & J. J. Preiss (Eds.), *Human organization research* [The analysis of qualitative field data] (pp. 279-289). Homewood, IL: Dorsey Press.
- Béguin, P. (2003). Design as a mutual learning process between users and designers. *Interacting with Computers*, 15(5), 709-730.
- Béguin, P. (2007). Taking activity into account during the design process. *Activités*, 4(2), 115-121.
- Béguin, P. (2009). When users and designers meet each other in the design process. In C. Owen, P. Béguin & G. Wackers (Eds.), *Risky work environments: Reappraising human work within fallible systems* (pp. 153-172). Padstow, Cornwall, UK: TJ International Ltd.
- Bertelsen, O. W. (2000). Design artefacts: Towards a design-oriented epistemology. *Scandinavian Journal of Information Systems*, 12(15), 27.
- Beyer, H., & Holtzblatt, K. (1997). *Contextual design: Defining customer-centered systems (interactive technologies)*. San Francisco; San Diego, CA; New York; Boston; London; Sydney, Australia; Tokyo: Morgan Kaufman.
- Biddle, R., Martin, A., & Noble, J. (2003). No name: Just notes on software reuse. *ACM SIGPLAN Notices*, 38(12), 76-96.
- Bødker, S. (1991). *Through the interface: A human activity approach to user interface design*. Hillside, NJ: Lawrence Erlbaum Associates, Inc.

- Bødker, S. (1996). Applying activity theory to video analysis: How to make sense of video data in human-computer interaction. In B. Nardi (Ed.), *Context and consciousness* (1st ed., pp. 145-174). Cambridge, MA: MIT Press.
- Bødker, S. (1998). Understanding representation in design. *Human-Computer Interaction*, 13(2), 107-125.
- Bødker, S., & Andersen, B. (2005). Complex mediation. *Journal of Human Computer Interaction*, 20(4), 353-402.
- Borman, L. (1996). SIGCHI: The early years. *SIGCHI Bulletin*, 28(1), 4-6.
- Bowen, J. (2006). *The Virtual Museum of Computing (VMoC)*. Retrieved February 18, 2006, from <http://icom.museum/vlmp/computing.html>
- Brereton, M. (2001). Drawing lessons in the design of tangible media from a study of interactions with mechanical products. *Second Australasian User Interface Conference (AUIC'01)*, Queensland, Australia. 124-131.
- British Computer Society. (2006a). *British Computer Society homepage*. Retrieved March 11, 2006, from <http://www.bcs.org/>
- British Computer Society. (2006b). *History of the British Computer Society*. Retrieved March 11, 2010, from <http://www.bcs.org/server.php?show=conWebDoc.1580>
- British Computer Society. (2006c). *Information Society examination board qualifications*. Retrieved March 11, 2010, from <http://www.bcs.org/server.php?show=nav.5732>
- Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20, 10-19.
- Brown, J. (1997). HCI and requirements engineering: Exploring human-computer interaction and software engineering methodologies for the creation of interactive software. *SIGCHI Bulletin*, 29(1), 32-35.
- Brown, J. (2003). Bringing multiple perspectives together to support multidisciplinary design teams. *Proceedings for the 5th World Congress on Management of Intellectual Capital and Innovation '03*, Hamilton, Ontario, Canada.
- Brown, J. (2005). A study of the professional identities of human-computer interaction specialists using grounded theory [Abstract]. *The Annual Convention of the Canadian Psychological Association*, 26(2a) 199.
- Brown, J., Biddle, R., & Lindgaard, G. (2008a). Stories, sketches, and lists: Developers and interaction designers interacting through artefacts. *Proceedings of Agile 2008*:

- Expanding Agile Horizons*, Toronto, Ontario, Canada. 39-50.
- Brown, J., Biddle, R., & Lindgaard, G. (2008b). Analyzing talk- and artefact-mediated interactions from a CHAT perspective [Abstract]. *Abstracts for the International Society for Cultural and Activity Research 2008*, page no. n/a.
- Brown, J., Graham, T. C. N., & Wright, T. (1998). The Vista environment for the coevolutionary design of user interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '98)*, Los Angeles, CA. 376-383.
- Brown, J., & Johnston, C. (2001a). Extending the value of prototypes with Panorama a tool to browse software artefacts. *Proceedings for the ACM/SIGCHINZ Symposium '01*, 15-20.
- Brown, J., & Johnston, C. (2001b). Panorama: A tool for browsing design artefacts. *IEEE/Proceedings of the Australasian Computer-Human Interaction Conference (OzCHI '01)*, Freemantle, Perth, Australia. 7-12.
- Brown, J., Lindgaard, G., & Biddle, R. (2007). Analyzing artefact-mediated interactions [Abstract]. *International Society for Theoretical Psychology Abstracts: Theoretical Psychology Beyond Borders: Transdisciplinarity and Internationalization*, 49.
- Brown, J., & Marshall, S. (1998). Sharing human-computer interaction and software engineering design artefacts. *IEEE/Australasian Computer-Human Interaction (OZCHI '98)*, Adelaide, Australia. 53-60.
- Brown, J., & Strickland, L. (2005a). Human-oriented technology from a philosophical and historical perspective. *History and Philosophy of Psychology Bulletin (HPPB)*, 17(2), 2-3.
- Brown, J., & Strickland, L. (2005b). The value of activity theory in directing designers to consider contextual issues in technological innovation and development [Abstract]. *International Society for Theoretical Psychology Abstracts (ISTP '05)*, 11.
- Brown, J., & Strickland, L. (2007). Activity theory in the field of human-computer interaction. In V. Van Deventer, M. Terre Blanche, E. Fourie & P. Segalo (Eds.), *Citizen City: Between constructing agent and constructed agency* (pp. 371-380). Concord, Ontario, Canada: Captus University Publications.
- Bucholtz, M. (2005). Identity and interaction: A sociocultural linguistic approach. *Discourse Studies*, 7(4-5), 585-614.
- Card, S. K., Newell, A., & Moran, T. P. (1983). *The Psychology of human-computer interaction*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

- Carroll, J. M. (Ed.). (1995). *Scenario-based design: Envisioning work and technology in system development*. New York: John Wiley.
- Carroll, J. M. (Ed.). (2003a). *HCI models, theories, and frameworks: Toward a multidisciplinary science*. San Francisco, CA: Morgan Kaufmann Publishers.
- Carroll, J. M. (2003b). Introduction: Toward a multidisciplinary science of human-computer interaction. In J. M. Carroll (Ed.), *HCI models, theories, and framework: Toward a multidisciplinary science* (pp. 1-9). San Francisco: Morgan Kaufmann Publishers.
- Carroll, J. M. (2009). *Encyclopedia entry on human computer interaction (HCI)*. Retrieved June 22, 2010, from http://www.interaction-design.org/encyclopedia/human_computer_interaction_hci.html
- Chaiklin, S. (2007). Modular or integrated?: An activity perspective for designing and evaluating computer-based systems. *International Journal of Human-Computer Interaction*, 22(1&2), 173-190.
- Clark, H. H. (1996). *Using language* (1st ed.). Cambridge, UK: Press Syndicate of the University of Cambridge.
- Cockburn, A. (2002). *Agile software development*. Boston: Addison Wesley.
- Cole, M. (1997). *Cultural psychology*. Cambridge, MA; London, UK: The Belknap Press of Harvard University Press.
- Cole, M., Levitin, K., & Luria, A. (2006). *The autobiography of Alexander Luria: A dialogue with the making of mind*. Mahwah, NJ: Lawrence Erlbaum Associates.
- Connors, M. (1999). *Engineers take Memorial University to court*. Retrieved March 11, 2010, from <http://www.peak.sfu.ca/the-peak/99-1/issue7/engineers.html>
- Constantine, L. L., & Lockwood, L. A. (2001). Structure and style in use cases for user interface design. In M. Van Harmelen, & S. Wilson (Eds.), *Object modeling and user interface design: Designing interactive systems* (pp. 245-279). Boston: Addison-Wesley.
- Constantine, L. L., & Lockwood, L. A. D. (2003). Usage-centered software engineering: An agile approach to integrating users, user interfaces, and usability into software engineering practice. *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, Portland, OR. 746-747.
- Coutaz, J. (1995). Evaluation techniques: Exploring the intersection of HCI and software engineering. *Lecture Notes in Computer Science Vol. 896* (pp. 35-48). New York: Springer-Verlag.

- Daniels, H. (2005a). Analysing institutional effects: First steps in the development of a language of description [Abstract]. *Abstracts for the International Society for Cultural Activity Research 2005*, page no. n/a.
- Daniels, H. (2005b). Subject position and discourse in activity theory [Abstract]. *Abstracts for International Society for Cultural Activity Research 2005*, page no. n/a.
- Daniels, H. (2008a). Activity theory and interventionist research. *Vygotsky and Research* (pp. 115-148). London; New York: Routledge.
- Daniels, H. (2008b). *Vygotsky and Research*. New York: Routledge.
- de Paula, M. G., Barbosa, S. D. J., & de Lucena, C. J. P. (2005). Conveying human-computer interaction concerns to software engineers through an interaction model. *CLIHIC '05: Proceedings of the 2005 Latin American Conference on Human-Computer Interaction*, Cuernavaca, Mexico. 109-119.
- de Souza, C. R. B., Froehlick, J., & Dourish, P. (2005). Seeking the source: Software source code as a social and technical artifact. *GROUP '05: Proceedings of the 2007 International ACM Conference on Supporting Group Work*, Sanibel Island, FL.
- de Souza, C. R. B., Quirk, S., Trainer, E., & Redmiles, D. (2007). Supporting collaborative software development through the visualization of socio-technical dependencies. *GROUP '07: Proceedings of the 2007 International ACM Conference on Supporting Group Work*, Sanibel Island, FL. 147-156.
- de Souza, C. R. B., Sharp, H., Singer, J., Cheng, L., & Venolia, G. (2009). Cooperative and human aspects of software engineering. *IEEE Software*, November/December, 17-19.
- DeGrace, P., & Stahl, H. L. (1998). *Wicked problems, righteous solutions: A catalog of modern engineering paradigms*. Englewood Cliffs, NJ: Prentice Hall.
- Denzin, N. K., & Lincoln, Y. S. (Eds.). (2005). *The SAGE handbook of qualitative research* (3rd ed.). Thousand Oaks, CA: Sage Publications.
- Dittrich, Y., Floyd, C., & Klischewski, R. (Eds.). (2002). *Social thinking: Software practice*. Boston: MIT Press.
- Dix, A., Finlay, J., Abowd, G., & Beale, R. (1998). *Human-computer interaction* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
- Downton, A. (1993). *Engineering the human-computer interface*. London: McGraw-Hill.
- Elliott, R., Fischer, C. T., & Rennie, D. L. (1999). Evolving guidelines for publication of qualitative research studies in psychology and related fields. *British Journal of*

- Clinical Psychology*, 38, 215-229.
- Engeström, Y. (1987). *Learning by expanding: An activity-theoretical approach to developmental research* (online version ed.). Helsinki, Finland: Orienta-Consultit Oy.
- Engeström, Y. (1995). Objects, contradictions and collaboration in medical cognition - An activity-theoretical perspective. *Artificial Intelligence in Medicine*, 7(5), 395-412.
- Engeström, Y. (1999a). Activity theory and transformation. In Y. Engeström, R. Miettinen & R. Punamèaki-Gitai (Eds.), *Perspectives on Activity Theory* (pp. 19-39). Cambridge, UK: Cambridge University Press.
- Engeström, Y. (1999b). Expansive visibilization of work: An activity-theoretical perspective. *Computer Supported Cooperative Work*, 8(1-2), 63-93.
- Engeström, Y. (1999c). Innovative learning in work teams: Analyzing cycles of knowledge creation in practice. In Y. Engeström, R. Miettinen & R. Punamèaki-Gitai (Eds.), *Perspectives on activity theory* (pp. 377-406). Cambridge, UK; New York: Cambridge University Press.
- Engeström, Y. (2000). Activity theory as a framework for analyzing and redesigning work. *Ergonomics*, 43(7), 960-974.
- Engeström, Y. (2001). Expansive learning at work: Toward an activity theoretical reconceptualization. *Journal of Education and Work*, 14(1), 133-156.
- Engeström, Y. (2002). *Center for activity theory and development work research: Research strategy for 2003-2005*. Retrieved from <http://www.edu.helsinki.fi/activity/>
- Engeström, Y. (2007). Putting Vygotsky to work: The change laboratory as an application of double stimulation. In H. Daniels, M. Cole & J. V. Wertsch (Eds.), *The Cambridge companion to Vygotsky* (pp. 363-383). New York: Cambridge University Press.
- Engeström, Y. (2008). *From teams to knots: Activity-theoretical studies of collaboration and learning at work*. Cambridge, UK; New York: Cambridge University Press.
- Engeström, Y., Brown, K., Christopher, L. C., & Gregory, J. (1997). Coordination, cooperation and communication in the courts: Expansive transitions in legal work. In M. Cole, Y. Engeström & O. Vasquez (Eds.), *Mind, culture and activity: Seminal papers from the Laboratory of Comparative Human Cognition* (pp. 369-389). Cambridge, UK: Cambridge University Press.
- Engeström, Y., & Engeström, R. (1986). Developmental work research: The approach and an application in cleaning work. *Nordisk Pedagogik*, 1, 2-15.

- Engeström, Y., Miettinen, R., & Punamèaki-Gitai, R. (Eds.). (1999). *Perspectives on activity theory*. Cambridge, UK; New York: Cambridge University Press.
- Engineering council UK. (2006). *Engineering Council of the United Kingdom*. Retrieved March 11, 2010, from <http://www.engc.org.uk/>
- Engineers Canada. (2004). *Canadian council of professional engineers accredited engineering programs*. Retrieved March 11, 2010, from http://www.engineerscanada.ca/e/pr_programs1.cfm
- Ensmenger, N. (2001). The 'question of professionalism' in the computer fields. *IEEE Annals of the History of Computing*, 23(4), 107-112.
- Ensmenger, N. (2004). Power to the people: Toward a social history of computing. *Annals of the History of Computing*, 26(1), 94-96.
- Ferre, X., Juristo, N., & Moreno, A. M. (2004). Improving software engineering practice with HCI aspects. *Lecture Notes in Computer Science*, 3026, 349-363.
- Fisher, G., Arias, E., Eden, H., Gorman, A., & Scharff, E. (2001). Transcending the individual human mind - creating shared understanding through collaborative design. In J. M. Carroll (Ed.), *Human-Computer Interaction in the new millennium* (pp. 347-372). Boston: Addison-Wesley.
- Fitzpatrick, G. (1998). *Locales framework: Understanding and designing for cooperative work*. (PhD, Department of Computer Science & Electrical Engineering, University of Queensland).
- Flowers, S. (1996). *Software failure, management failure: Amazing stories and cautionary tales*. New York: John Wiley and Sons, Inc.
- Glaser, B. G. (1965). The constant comparative method of qualitative analysis. *Social Problems*, 12(4), 436-445.
- Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Rutgers The State University, NJ: Aldine Transaction, A Division of Transaction Publishers.
- Glass, R. L. (1996). The relationship between theory and practice in software engineering. *Communications of the ACM*, 39(11), 11-13.
- Glass, R. L. (1999). *Computing calamities: Lessons learned from products, projects, and companies that failed*. Upper Saddle River, NJ: Prentice Hall,.
- Glelen, U. P., & Jesmaridian, S. S. (1999). Lev S. Vygotsky: The man and the era. *International Journal of Group Tensions*, 28(3), 273-301.

- Grad, B., & Johnson, L. (2002). The start of the software products industry. *IEEE Annals of the History of Computing*, 24(1), 3-4.
- Greenbaum, J. M., & Kyng, M. (Eds.). (1991). *Design at work: Cooperative design of computer systems*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Grudin, J. (1990). The computer reaches out: The historical continuity of interface design. *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Seattle, WA. 261-268.
- Grudin, J. (1993). Interface: An evolving concept. *Communications of the ACM*, 36(4), 110-119.
- Grudin, J. (2004). Crossing the divide. *ACM Transactions on Computer-Human Interaction*, 11(1), 1-25.
- Grudin, J. (2005). Three faces of human-computer interaction. *IEEE Annals of the History of Computing*, 27(4), 46-62.
- Guba, E. G., & Lincoln, Y. S. (1989). *Fourth generation evaluation*. Newbury Park, CA: Sage.
- Guba, E. G., & Lincoln, Y. S. (2000). Paradigmatic controversies, contradictions, and emerging confluences. In N. K. Denzin, & Y. S. Lincoln (Eds.), *The Sage handbook of qualitative research* (3rd ed., pp. 191-215). Thousand Oaks, CA; New Delhi, India; London: Sage.
- Harning, M. B. (2004). *Bridging the SE and HCI communities*. Retrieved August, 26, 2006, from <http://www.se-hci.org/bridging/>
- Harper, D. (2005). What's new visually? In N. K. Denzin, & Y. S. Lincoln (Eds.), *The SAGE handbook of qualitative research* (3rd ed., pp. 747-762). Thousand Oaks, CA: Sage Publications.
- Harrison, W., Ossher, H., & Tarr, P. (2000). Software engineering tools and environments: A roadmap. *ICSE '00: Proceedings of the Conference on the Future of Software Engineering*, Limerick, Ireland. 261-277.
- Hewett, T. T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., & Verplank, W. (1992). *ACM SIGCHI curricula for human-computer interaction* (An ACM SIGCHI committee report of the ACM SIGCHI Curriculum Development Group No. Not available). New York: ACM.
- Holland, D., & Lachicotte, W. (2007). Vygotsky, Mead and the new sociocultural studies of identity. In H. Daniels, M. Cole & J. V. Wertsch (Eds.), *The Cambridge companion to Vygotsky* (pp. 101-135). New York: Cambridge University Press.

- John, B. E., Bass, L., & Adams, R. J. (2003). Communication across the HCI/SE divide: ISO13407 and the rational unified process. *Proceedings of HCI International*, Crete.
- John-Steiner, V. P. (2007). Vygotsky on thinking and speaking. In H. Daniels, M. Cole & J. V. Wertsch (Eds.), *The Cambridge companion to Vygotsky* (pp. 136-154). New York: Cambridge University Press.
- Jordan, B. (1996). Ethnographic workplace studies and computer supported cooperative work. In D. Shapiro, M. Tauber & R. Tranmüller (Eds.), *The Design of Computer-Supported Cooperative Work and Groupware Systems* (pp. 17-42). Amsterdam, The Netherlands: North Holland/Elsevier Science.
- Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *The Journal of the Learning Sciences*, 4(1), 39-103.
- Kaptelinin, V. (1996). Activity theory: Implications for Human-Computer Interaction. In B. Nardi (Ed.), *Context and consciousness: Activity Theory and Human-Computer Interaction* (pp. 103-116). Boston: MIT Press.
- Kaptelinin, V. (2005). The object of activity: Making sense of the sense-maker. *Mind, Culture and Activity*, 12(1), 4-18.
- Kaptelinin, V., & Cole, M. (1997). Individual and collective activities in educational computer game playing. Paper presented at the *Computer Supported Cooperative Learning '97*, 136-141. Retrieved from <http://lchc.ucsd.edu/People/MCole/Activities.html>
- Keet, E. E. (2004). A personal recollection of software's early days (1960-1979): Part 1. *Annals of the History of Computing*, 26(4), 46-61.
- Kerosuo, H. (2001). 'Boundary Encounters' as a place for learning and development at work. *Outlines: Critical Social Studies*, 3(1), 53-65.
- Kerosuo, H., & Engeström, Y. (2003). Boundary crossing and learning in creation of new work practice. *Journal of Workplace Learning*, 15(7/8), 345-351.
- Kruchten, P. (2010). Certification 1, 2, 3. *IEEE Software*, 27(3), 92-94.
- Kruchten, P. (2004). *The rational unified process: An introduction* (3rd ed.). Boston: Addison-Wesley.
- Kuutti, K. (1996). Activity theory as a potential framework for Human-Computer Interaction research. In B. Nardi (Ed.), *Context and consciousness: Activity theory and Human-Computer Interaction* (pp. 17-44). Cambridge, MA; London: MIT Press.
- Kuutti, K. (1999). Activity theory, transformation of work and information systems design. In Y. Engeström, R. Miettinen & R. Punamaki (Eds.), *Perspectives on*

- activity theory* (pp. 360-376). Cambridge, UK: Cambridge University Press.
- Laurel, B. (1991). *Computers as theatre*. Reading, MA: Addison-Wesley.
- Lemke, J. L. Identity, development, and desire: Critical questions. (April, 2003)
Unpublished manuscript. Paper presented at the meeting of the American Educational Research Association, Chicago, IL. Available from http://www-personal.umich.edu/~jaylemke/papers/Identity/identity_aera_2003.htm
- Leontiev, A. N. (1972). *Problems of the development of the mind* [Problemy razvitiya psikhiki] (3rd ed.). Moscow: MGU.
- Leontiev, A. N. (1977). Activity and consciousness. *Philosophy in the USSR: Problems of dialectical materialism* (pp. 180-202). Moscow: Progress Publishers.
- Leontiev, A. N. (1978a). Activity and consciousness. *Activity, consciousness and personality* (pp. 75-96). Englewood Cliffs, NJ: Prentice Hall, Inc.
- Leontiev, A. N. (1978b). Activity and personality. *Activity, consciousness and personality* (pp. 97-140). Englewood Cliffs, NJ: Prentice Hall, Inc.
- Leontiev, A. N. (1978c). *Activity, consciousness and personality*. Englewood Cliffs, NJ: Prentice Hall, Inc.
- Leontiev, A. N. (1978d). The problem of activity and psychology. *Activity, consciousness and personality* (pp. 45-74). Englewood Cliffs, NJ: Prentice Hall, Inc.
- Leontiev, A. N. (1978e). Psychic reflection. *Activity, consciousness and personality* (pp. 29-44). Englewood Cliffs, NJ: Prentice Hall, Inc.
- Leontiev, A. N. (1981). *Problems of the development of the mind* (M. Kopylova Trans.). Moscow: Progress Publishers.
- Leontiev, A. N. (1997). On Vygotsky's creative development. In R. W. Rieber, & J. Wollock (Eds.), *The collected works of L.S. Vygotsky: Problems of the theory and history of psychology* (pp. 9-35). New York: Plenum Press.
- Lindgaard, G. (2006). What's IFIP and why does it matter? *HOT Topics*, 5(7) Retrieved from <http://hot.carleton.ca/hot-topics/articles/whats-ifip>
- Lindgaard, G., Havloujian, J., Chessari, J., & Bednall, E. S. (1993). *The human factors kit* (Technical Report No. 93/112). Australia: Telstra.
- Luborsky, M. R., & Rubinstein, R. L. (1995). Sampling in qualitative research: Rationale, issues, and methods. *Research on Aging*, 17, 89-113.

- Mahoney, M. S. (1988). The history of computing in the history of technology. The *Annals of the History of Computing*, 10(2), 113-125.
- Mahoney, M. S. (1990). The roots of software engineering. *CWI Quarterly*, 3(4), 325-334.
- Mahoney, M. S. (2000). Probing the elephant: How do the parts fit together? In U. Hashagen, R. Keil-Slawik & A. Norberg (Eds.), *History of computing: Software issues* (pp. 285-291). Berlin, Germany: Springer Verlag.
- Mahoney, M. S. (2002). Software as science - Science as software. In U. Hashagen, R. Keil-Slawik & A. Norberg (Eds.), *History of Computing: Software Issues* (pp. 25-48). Berlin, Germany: Springer Verlag.
- Mahoney, M. S. (2004). Finding a history for software engineering. *Annals of the History of Computing*, 26(1), 8-19.
- Mahoney, M. S. (2005). The histories of computing(s). *Interdisciplinary Science Reviews*, 30(2), 119-135.
- Mallalieu, G., Harvey, C., & Hardy, C. (1999). The wicked relationship between organizations and information technology. *Journal of End User Computing*, 11(0), 40-59.
- McIlroy, M. D. (1976). Mass-produced software components. Garmisch-Partenkirchen. 138-150.
- Mommel, T., Vanderdonckt, J., & Reiterer, H. (2008). Multi-fidelity user interface specifications. In T. C. N. Graham, & P. Palanque (Eds.), *Interactive Systems, Design, Specification, and Verification* (pp. 43-57). Berlin/Heidelberg, Germany: Springer Verlag.
- Mercer, N. (1995). *The guided construction of knowledge: Talk amongst teachers and learners*. Clevedon, Avon, England; Philadelphia: Multilingual Matters.
- Mercer, N., & Staarman, J. K. (2005). Dialogue and activity: A methodological gap that needs to be filled [Abstract]. *Abstracts for International Society for Cultural Activity Research 2005*, page no. n/a.
- Miettinen, R., & Hasu, M. (2002). Articulating user needs in collaborative design: Towards an activity-theoretical approach. *Computer Supported Collaborative Work*, 11(1-2) 129-151. New Orleans, LA: SpringerLink.
- Morrow, S. L. (2005). Quality and trustworthiness in qualitative research in counseling psychology. *Journal of Counseling Psychology*, 52(2), 250-260.

- Myers, B., Hudson, S. E., & Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, 7(1), 3-28.
- National Society of Professional Engineers. (2006). *Task force report and recommendation for computer and software engineering licensure path*. Alexandria, VA: National Society of Professional Engineers. Retrieved from www.nspe.org/resources/documents/pei/comp-sw_eng_for_lqp.doc
- Naur, P., & Randell, B. (1994). *Software engineering: Report on a conference sponsored by the NATO science committee* (Report on a conference No. n/a). Garmisch Germany: NATO Science Committee.
- Newman, F., & Holzman, L. (1993). Practice: Vygotsky's tool-and-result methodology and psychology. In F. Newman, & L. Holzman (Eds.), *Lev Vygotsky: Revolutionary scientist* (pp. 32-54). London; New York: Routledge.
- Nielsen, J. (1994). *Usability engineering*. San Francisco, CA: Morgan Kaufmann.
- Noble, J., & Biddle, R. (2006). Postmodern prospects for conceptual modelling. Paper presented at the *Proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCMM2006)*, Hobart, Australia. 11-20.
- Noble, J., & Biddle, R. (2002). Notes on postmodern programming. *Proceedings of the Onward Track at Oopsla '02*, Seattle, WA. 49-71.
- Noble, J., & Biddle, R. (2004). Notes on notes on postmodern programming. *Special Interest Group for Programming Languages Notices*, 39(12), 40-56.
- Norman, D. A., & Draper, S. W. (1986). *User centered system design: New perspectives on human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
- Owen, C., Béguin, P., & Wackers, G. (Eds.). (2009). *Risky work environments: Reappraising human work within fallible systems*. Padstow, Cornwall: TJ International Ltd.
- Patton, J. (2005). Finding the forest in the trees. *OOPSLA Companion*, 266-274.
- Pfleeger, S., Lawrence. (1998). *Software engineering: Theory and practice* (1st ed.). Upper Saddle River, NJ: Prentice-Hall.
- Ponterotto, J. G. (2005). Qualitative research in counseling psychology: A primer on research paradigms and philosophy of science. *Journal of Counseling Psychology*, 52(2), 126-136.
- Ponterotto, J. G. (2007). Effectively communicating qualitative research. *The Counseling Psychologist*, 35(3), 404-430.

- Pressman, R. S. (1992). *Software engineering: A practitioner's approach* (3rd ed.). New York: McGraw-Hill.
- Professional Engineers of Ontario. (2004). *Briefing Report on Software Practice*. Ontario, Canada: Professional Engineers of Ontario. Retrieved from http://www.peo.on.ca/enforcement/Software_Briefing_Paper2004.pdf
- Raccoon, L. B. S. (1997). Fifty years of progress in software engineering. *SIGSOFT Software Engineering Notes*, 22(1), 88-104.
- Radle, K., & Young, S. (2001). Partnering usability with development: How three organizations succeeded. *IEEE Software*, January/February, 38-45.
- Randell, B. (1979). Software Engineering in 1968. *Proceedings of the 4th International Conference on Software Engineering*, 1-10.
- Randell, B. (1996). *The 1968/69 NATO Software Engineering Reports* (Dagstuhl-Seminar No. 9635). Dagstuhl, Germany: NATO.
- Ratner, C. (1991). *Vygotsky's sociohistorical psychology and its contemporary applications*. New York, NY: Plenum Press.
- Rittel, H., & Webber, M. (1973). Dilemmas in a general theory of planning. *Policy Sciences*, 4, 155-159.
- Robertson, T. (1997). Designing over distance: A study of cooperative work, embodied cognition and technology to enable remote collaboration. (PhD, School of Computer Science, University of Technology Sydney). Retrieved from <http://research.it.uts.edu.au/idwop/downloads/RobertsonThesis.pdf>
- Robertson, T. (2000). Building bridges: Negotiating the gap between work practice and technology design. *International Journal of Human-Computer Studies*, 53, 121-146.
- Robinson, H., Hall, P., Hovenden, F., & Rachel, J. (1998). Postmodern software development. *The Computer Journal*, 41(6), 363-375.
- Rosson, M. B., & Carroll, J. M. (2001). Scenarios, objects, and points-of-view in user interface design. In M. Van Harmelen, & S. Wilson (Eds.), *Object modeling and user interface design: Designing interactive systems* (pp. 39-69). Boston: Addison-Wesley.
- Roth, W. (2005). *Doing qualitative research: Praxis of method*. Rotterdam, The Netherlands: Sense Publishers.
- Rowlands, J. (1999, September/October). "Wild West" or maturing discipline? A look at standards and regulation in software engineering. *Engineering Dimensions*, 20, 30-

33.

- Rubin, H. J., & Rubin, I. S. (1995). Building conversational partnerships. In H. J. Rubin, & I. S. Rubin (Eds.), *Qualitative interviewing: The art of hearing data* (pp. 93-121). Thousand Oaks, CA: Sage.
- Ruhleder, K., & Jordan, B. (2001). Co-constructing non-mutual realities: Delay-generated trouble in distributed interaction. *Computer Supported Cooperative Work*, 10(1), 113-138.
- Sammet, J. E. (1969). *Programming languages: History and fundamentals* (1st ed.). Englewood Cliffs, NJ: Prentice-Hall.
- Santamaria, A., Cubero, M., & de la Mata, M. L. (2010). Thinking as action: Theoretical and methodological requirements for cultural psychology. *Theory & Psychology*, 20(1), 76-101.
- Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. New York: Basic Books.
- Schwaber, K. (2003). *Agile project management with Scrum*. Redmond, WA: Microsoft Press.
- Seale, C. (1999a). Converging on a point? *The quality of qualitative research* (pp. 52-72). Thousand Oaks, CA: Sage.
- Seale, C. (1999b). Criteria for the evaluation of qualitative research papers. *The quality of qualitative research* (pp. 189-192). Thousand Oaks, CA: Sage.
- Seffah, A., Djouab, R., & Antunes, H. (2001). Comparing and reconciling usability-centered and use case-driven requirements engineering processes. *AUIC '01: Proceedings of the 2nd Australasian Conference on User Interfaces*, Queensland, Australia. 132-139.
- Seffah, A., & Metzker, E. (2004). The obstacles and myths of usability and software engineering. *Communications of the ACM*, 47(12), 71-76.
- Shapiro, S. (1997). Splitting the difference: The historical necessity of synthesis in software engineering. *Annals of the History of Computing*, 19(1), 20-54.
- Sharp, H., Robinson, H., & Petre, M. (2009). The role of physical artefacts in agile software development: Two complementary perspectives. *Interacting with Computers*, 21(1-2), 108-116.
- Sharp, H., Rogers, Y., & Preece, J. (2002). *Interaction design: Beyond human-computer interaction*. New York: John Wiley and Sons, Inc.

- Shneiderman, B. (1987). *Designing the user interface: strategies for effective human-computer interaction*. Reading, MA: Addison-Wesley.
- Shneiderman, B., & Plaisant, C. (2004). *Designing the user interface: Strategies for effective human-computer interaction* (4th ed.). Reading, MA: Addison-Wesley.
- Silverman, D. (2005). Quality in Qualitative Research. *Doing Qualitative Research* (2nd ed., pp. 209-223). Thousand Oaks, CA; New Delhi, India: Sage.
- Simon, H. (1996). *The sciences of the artificial* (3rd ed.). Cambridge, MA: MIT Press.
- Smith, S. L., & Mosier, J. N. (1986). *Guidelines for designing user interface software* (1st ed.). Springfield, VA: National Technical Information Service.
- Smith, J. A. (Ed.). (1995). *Semi-structured interviewing and qualitative analysis*. Thousand Oaks, CA: Sage.
- Sousa, K., Furtado, E., & Mendonca, H. (2005). Upi -- A software development process aiming at usability, productivity and integration. *Proceedings of CLIHC '05*, Cuernavaca, Mexico. 76-87.
- Spinuzzi, C. (2003). Compound mediation in software development: Using genre ecologies to study textual artifacts. In C. Bazerman, & D. R. Russell (Eds.), *Writing selves and societies: Research from activity perspectives* (pp. 97-124). Fort Collins, Colorado: The WAC Clearinghouse and Mind, Culture, and Activity. Retrieved from http://wac.colostate.edu/books/selves_societies/
- Stake, R. E. (2005). Qualitative case studies. In N. K. Denzin, & Y. S. Lincoln (Eds.), *The SAGE handbook of qualitative research* (3rd ed., pp. 443-466). Thousand Oaks, CA: Sage Publications.
- Strauss, A. L. (1993). *Continual permutations of actions*. New York: Aldine de Gruyter.
- Strauss, A. L., & Corbin, J. M. (1998a). *Basics of qualitative research: Techniques and procedure for developing grounded theory*. Thousand Oaks, CA: Sage.
- Strauss, A. L., & Corbin, J. M. (1998b). Criteria for evaluation. *Basics of qualitative research: Techniques and procedure for developing grounded theory* (pp. 265-274). Thousand Oaks, CA: Sage.
- Strauss, A. L., & Corbin, J. M. (1998c). Theoretical Sampling. *Basics of qualitative research: Techniques and procedure for developing grounded theory* (pp. 201-215). Thousand Oaks, CA: Sage.
- Strauss, A. L., & Corbin, J. M. (1998d). *Grounded theory in practice*. Thousand Oaks, CA: Sage.

- Suchman, L. (1987). *Plans and situated action: The problem of human-machine communication*. New York: Cambridge University Press.
- Suchman, L. (1995). Making work visible. *Communications of the ACM*, 38(9), 56-64.
- Suchman, L. (2000). Embodied practices of engineering work. *Journal of Mind, Culture and Activity*, 7(1&2), 4-18.
- Susi, T., & Ziemke, T. (2001). Social cognition, artefacts, and stigmergy: A comparative analysis of theoretical frameworks for the understanding of artefact-mediated collaborative activity. *Cognitive Systems Research*, 2(4), 263-290.
- Swidler, A. (1986). Culture in action: Symbols and strategies. *American Sociological Review*, 51(2), 273-286.
- Tatar, D. (1989). Using video-based observation to shape the design of a new technology. *Bulletin of the Special Interest Group in Computer-Human Interaction*, 21(2), 108-111.
- Terre Blanche, M., & Durrheim, K. (1999). *Research in practice: Applied methods for the social sciences*. Lansdowne, South Africa: Juta Academic.
- Tuikka, T. (2002). Remote concept design from an activity theory perspective. *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, New Orleans, LA. 186-195.
- Tuikka, T., & Kuutti, K. (2000). Making new design ideas more concrete. *Knowledge-Based Systems*, 13, 395-402.
- Tuikka, T., & Kuutti, K. (2001). *Thinking together in concept design for future products: Emergent features for computer support*. Berlin, Germany: Springer Verlag.
- Turner, P., & Turner, S. (2001). A web of contradictions. *Interacting with Computers*, 14(1), 1-14.
- UPA Board of Directors. (2002). *UPA board of directors position statement on certification for usability professionals*. Unpublished manuscript. Retrieved June 22, 2010, from http://www.upassoc.org/about_upa/for_the_press/press_releases/press.020709.html
- van der Riet, M. (2009). The production of context: Using activity theory to understand behavior change in response to HIV and AIDS. (PhD, University of KwaZulu-Natal). Retrieved from http://researchspace.ukzn.ac.za/xmlui/bitstream/handle/10413/704/Van%20der%20Riet_M_2009.pdf?sequence=1

- Van Harmelen, M., & Wilson, S. (Eds.). (2001). *Object modeling and user interface design: Designing interactive systems*. Boston: Addison-Wesley.
- Vygotsky, L. S. (1978a). Internalization of higher psychological functions. In M. Cole, V. P. John-Steiner, S. Scribner & E. Souberman (Eds.), *Mind in society* (pp. 52-57). Cambridge, MA; London: Harvard University Press.
- Vygotsky, L. S. (1978b). In Cole M., John-Steiner V. P., Scribner S. and Souberman E. (Eds.), *Mind in society*. Cambridge, MA; London: Harvard University Press.
- Vygotsky, L. S. (1978c). Problems of method. In M. Cole, V. P. John-Steiner, S. Scribner & E. Souberman (Eds.), *Mind in society* (pp. 58-75). Cambridge, MA; London: Harvard University Press.
- Vygotsky, L. S. (1986). In Kozulin A. (Ed.), *Thought and language* (A. Kozulin Trans.). Cambridge, MA: MIT Press.
- Vygotsky, L. S. (1994). The methods of reflexological and psychological investigation. In R. van der Veer, & J. Valsiner (Eds.), *The Vygotsky reader* (pp. 27-45)
- Vygotsky, L. S. (1997). The historical meaning of the crisis in psychology: A methodological investigation. In R. W. Rieber, & J. Wollock (Eds.), *The collected works of L.S.Vygotsky: Problems of the theory and history of psychology* (pp. 233-345). New York: Plenum Press.
- Wardekker, W. L. (2000). Criteria for the quality of inquiry. *Mind, Culture, and Activity: An International Journal*, 7(4), 259-272.
- Warr, A., & O'Neill, E. (2006). The effect of group composition on divergent thinking in an interaction design activity. *Proceedings of the 6th Conference on Designing Interactive Systems*, University Park, PA. 122-132.
- Warr, A., & O'Neill, E. (2007). Tool support for creativity using externalizations. *Proceedings of the 6th ACM SIGCHI Conference on Creativity and Cognition*, Washington, DC. 127-136.
- Wartofsky, M. (1973). Perception, representation, and the forms of action: Toward an historical epistemology. *Models* (pp. 188-210). Dordrecht, Holland: D. Reidel.
- Wells, G. (2002). The role of dialogue in activity theory. *Mind, Culture, and Activity*, 9(1), 43-66.
- Wertsch, J. V. (1985). *Vygotsky and the social formation of mind*. Cambridge MA; London: Harvard University Press.

- Wertsch, J. V. (1991). *Voices of the mind: A sociocultural approach to mediated action*. Cambridge, MA; London: Harvard University Press.
- Wertsch, J. V. (1993). The multivoicedness of meaning. *Voices of the mind: A sociocultural approach to mediated action* (pp. 67-90). Cambridge, MA; London: Harvard University Press.
- Wertsch, J. V. (1998). *Mind as action*. New York: Oxford University Press.
- Wikinfo. (2006). *Software engineering*. Retrieved June 26, 2010, from http://www.wikinfo.org/index.php/Software_engineering
- Wilson, D. D., Cole, M., Nixon, A. S., Nocon, H., Gordon, V., Jackson, T., Garía, O. L., & Minami, Y. (2010). Qualitative research: Cultural-historical activity theory. In G. McGaw, P. Peterson & E. Baker (Eds.), *International encyclopedia of education* (3rd ed.,). New York: Elsevier.
- Yaroshevsky, M. G. (1996). Marxism in Soviet psychology: The social role of Russian science. In V. A. Koltsova, Y. N. Oleinik, A. R. Gilgen & C. K. Gilgen (Eds.), *Post-Soviet perspectives on Russian psychology* (pp. 161-186). Westport, CT: Greenwood Publishing Group, Inc.
- Zachry, M. (2001). Constructing usable documentation: A study of communicative practices and the early uses of mainframe computing in industry. *Journal of Technical Writing and Communication*, 31(1), 61-76.

Appendix A: Glossary of Terms

Agile: An approach to software development and management that aims to be responsive to changing requirements.

Aims: Refers to all motives, objectives and outcomes of an activity. Examples include implementability, ease of use, and usefulness.

Activity: Activity is directed toward an object, which links the activity to human needs. Activities can be individual or collective. Other attributes of activities are their motives, form, methods, emotional intensity, time requirements, space requirements and the physiological mechanisms of individuals engaged in the activity. According to Engeström, the interrelated elements of an activity system are: subject, tools, object/objectives, rules, community, "division of labour", motive, and outcome.

Activity theory: In this dissertation this is used to refer to a psychological framework (not a theory) with its roots in Vygotsky's cultural-historical psychology founded by Alexei N. Leontiev and Alexander Luria. This approach understands the human mind as internal and external mediated-actions that constitute complex, organized, socially-situated phenomena called activities.

Agenda: Mahoney (2002b) describes agenda as what a discipline's practitioners agree ought to be done, a consensus concerning the problems of the field, their order of importance or priority, the means of solving them, and what constitutes a solution.

Artefact: "An artifact is an aspect of the material world that has been modified over the history of its incorporation in goal directed human action. By virtue of the changes wrought in the process of their creation and use, artifacts are simultaneously ideal (conceptual) and material. ... They are manufactured in the process of goal directed human actions. They are ideal in that their material form has been shaped by their participation in the interactions of which they were previously a part and which they mediate in the present. ... Defined in this manner, the properties of artifacts apply with equal force whether one is considering language/speech or the more usually noted forms of artifacts such as tables and knives which constitute material culture. What differentiates the word 'table' from an actual table is the relative prominence of their material and ideal aspects and the kinds of coordinations they afford. No word exists apart from its material instantiation (as a configuration of sound waves, hand movements, writing, or neuronal activity), whereas every table embodies an order imposed by thinking human beings." (Cole, 1997)

Concept design: An early stage of software production work where the product proceeds from a vague idea to a well-developed one with important elements more or less fully specified and possibly even implemented.

Consciousness: An aspect of psychic reflection. A higher-level form of the psyche. Consciousness governs goal-oriented actions.

Constant comparative method: This method groups quotes based on comparisons of "complete verbal expressions of an attitude or complete acts by an individual or group."

Culture: A "system of artefacts" that "comes into being whenever people engage in joint activity over a period of time" (Cole, 2004, p. 301). With this definition of

culture, its meaning can be narrow (referring to the culture of a particular activity) or broad (referring to the collective activities of a body of people).

Cultural-historical activity theory (CHAT): A branch of cultural-historical psychology grounded in Vygotsky and Leontiev's work particularly, and which is particularly interested in historicity, activity, action, and culture.

Cultural-historical psychology: One of a number of psychologies in the broader field of psychology. It is a dynamic and diffuse framework that has been variously interpreted by researchers and practitioners. It emerged around 1920 in the early days of the Soviet Union as a reaction to other psychologies through the efforts of Vygotsky and colleagues. Cultural-historical psychology drew attention to human behaviours as a means to understand the mind in action, drawing heavily on the writings of German philosophers, particularly Hegel and Marx.

Designer: Designers in this research were individuals who were specifically responsible for both the visual and behavioural aspects of the user interface. These designers referred to themselves as human-computer interaction specialists, user interface designers, user experience designers, human factors specialists, graphic designers, content designers, or learning designers.

Design terrain: Addresses the look and feel and use of the user interface from the end user's perspective.

Developer: The software developers in this research were specialists trained, usually through computer science degrees, to create reliable, industry-standard software applications. All were responsible for writing code to implement the appearance and behaviour of the user interface and therefore worked closely with designers. Most also had other responsibilities including writing the program's logic, determining and arranging the software's components, writing code to connect to other software, or writing code to manage external data.

Division of labour: The roles within the community.

Focus-On chart: A charting method created to depict an interrelated system of activities and individuals and artefacts that are used in the software creation activity to achieve real or perceived aims in other activities.

History: There are two meanings for this word. One is the traditional understanding of a history as a chronological record of events; however, in activity theory, history, also means "process."

HCI: human-computer interaction. A discipline.

Implementation terrain: Issues related to the implementation of the user interface and corresponding back-end software.

Interaction: A brief periods in time during which two or more individuals purposefully engage with one another. These are frequently mediated.

Interaction designer: A content designer, learning designer, user interaction designer, user interface designer, interaction design team lead, senior user interaction designer, user experience designer, graphical user interface (GUI) designer or information technology (IT) systems analyst.

Interactional identity: How a team member sees himself or herself in interaction with others on his team. An interactional identity augments an individual's work identity and role and indicates how an individual relates to others on the team. It is also an extension of personality, as understood in cultural-historical activity theory. An Interactional Identity appears to be more closely determined by the

individual themselves as a response to who they are, the activity they are engaged in, and their past experiences. An Interactional Identity provides the sense of the approach an individual brings to their work. Hence, an Interactional Identity represents an approach to interacting that is adopted by an individual and that is used in interactions, including artefact-mediated interactions, with other team members.

Look and feel: The appearance and behavior of the user interface.

Meaning: A unit of analysis used by Vygotsky that encompasses both the individual and the social.

Modernism: A rational, ordered view of the world.

Motive: The reason for the object of activity.

Object: An element/elements that is/are transformed by an action or an activity.

Objectives: Desirable social and cultural attributes of an object.

Operations: Automatic actions.

Outcome: The result or output of the activity.

Postmodernism: The qualities of a postmodern view of the world are: “indeterminacies, fragmentation, decanonization, self-less-ness and depth-less-ness, [the] unrepresentable and unrepresentable, irony, hybridization, carnivalization, performance and participation, constructionism and immanence” (Huyssen as cited in Robinson, 1998).

Phase: A sequence of actions or interactions that are directed toward and/or mediated by a single set of design artefacts.

Reflection: A human mechanism for perceiving and understanding the world.

Role: Each role determines a particular responsibility vis-à-vis the object of the activity.

Rules: These mediate between a subject and the community.

Scrum: A particular agile approach to software development that emphasizes the management of software projects.

SIGCHI: Special Interest Group in Computer-Human Interaction, a body of researchers and practitioners whose aim is to advance the discipline and practice of human-computer interaction (HCI).

Silver bullets: Single solutions to large problems.

Software developer: A game developer, software developer, software designer, lead GUI designer, learning technology consultant, founding developer, software architect or Java developer.

Software engineer: A term popularized at a conference of software developers in Garmisch in 1968.

Software gap: The difference between the capability of the hardware and the software.

Software production: A view of software creation that compares it to process that might occur in a production plant.

Sprint: In Scrum this refers to a period of work spanning weeks, at the end of which specific items have been removed from the sprint backlog.

Strauss' theory of action: A framework underlying one form of grounded theory. The major concepts of this framework are actions, interactions and social worlds.

Subjects: The term used by activity theorists to refer to the individual or individuals engaged in an activity. In a research context, subjects are referred to as participants.

Theory or theoretical foundation: In computer science “theory” means a mathematical proof.

Tools: Tools mediate between a subject and object.

Unconsciousness/nonconsciousness: A lower level form of the psyche. Governs conditionally-determined actions called operations.

Units of analysis used in this dissertation: A meeting, a phase within a meeting (called a moment by Leontiev), mediated interactions, automatic uses of artefacts.

User interface, user, and designer: Terms that originated from a software engineering perspective, but which have become popular in the SIGCHI community to refer to the look and feel of an application, the individual who uses the application, and the person who is responsible for determining the look and feel of the application.

Appendix B: Demographics questions

Name?

Any preferred pseudonym?

Number of years with the company you are working for now?

Number of years working in your current professional capacity?

Number of years working in your current profession in any capacity?

Major role at your company?

Educational background (e.g., computer science, business, human factors, psychology, design, ...) Use multiple terms if these apply.

Relationship with HCI Group at your company?

Relationship with Software Development Group at your company?

Are you willing to provide feedback on my interpretations of the data?

Do you want a summary of the research findings?

Appendix C: Interview questions—Artefacts used in software-creation work

Briefly describe the project you are currently working on.

Is it a typical project? How would you characterize it?

What role do you and others play in the design activity on this particular project?

Tell me more specifically about the roles and responsibilities of the software developers and the designers on this project.

What are the individual goals of people involved in this project?

What is the collective goal of the designers on this project?

What is the collective goal of the developers on this project?

How can you tell when a goal has been accomplished?

How do you measure the success of your project in the end?

Are there tensions between any of the goals you've identified; are there occasions when satisfying one project goal may mean that another goal is sacrificed?

Are there tensions between the goals of the designers and the goals of the developers on this project?

What tensions might there be with goals from other activities you or others on the design team are involved in?

How could you / do you resolve these tensions?

What artefacts does your group use when designing?

How does your design team use artefacts in your workplace, both over time, and in the way you arrange them physically or virtually?

Describe the ways artefacts are used to accomplish design work in your project.

Who used each artefact and when? Elaborate especially on any artefacts that are used by both developers and designers.

Of the shared artefacts, were they efficient (increase productivity), effective (lead to the desired outcome) or reliable (don't cause errors or problems).

Are some artefacts useful to certain types of problems, but not others? Or certain scales of problems and not others?

Does everybody use the artefacts we've talked about in the same way?

Are there artefacts you'd like to be using, but aren't?

How do artefacts help you to interact with other members of your design team, especially with the designer or developer?

Do you ever develop artefacts specifically to communicate with fellow designers or developers?

Do you find that others communicate with you through their artefacts?

In your opinion, is this effective?

What are the rules, norms and procedures influencing how you work?

Appendix D: Interview questions—Software-creation work over time

Briefly describe your current workflow with regard to software creation on project X.

How do artefacts fit into your workflow?

Do the artefacts you use affect how you structure design work?

Has design work always been conducted in the same way at your company?

Tell me about large shifts in the organization of the design work at your company and identify the groups involved.

What prompted these changes?

In the past, how did groups communicate with each other to accomplish design work?

What were the goals of the various groups?

How could you tell when a group achieved its goal?

What tools were used by the various groups and what were the purposes of the tools?

Describe the roles people played within the various groups.

What rules determined how the groups related?

Were any artefacts passed between the groups?

What were the reasons (positive or negative) for the major changes in the way design work was done?

Have your goals or the goals of other groups or the design group as a whole changed over time?

What was the best project you ever worked on and why?

Appendix E: Applying cultural-historically influenced grounded theory

	Step	Key outcome in this step
1	Open Coding	Identity activity-theoretical influenced concepts.
2	Selective Coding	Evaluate concepts of relevance to research questions.
3	“Memoing” by Participant	Process quotations for each relevant concept, for each participant, in a memo.
4	Summarizing Memo Findings	Summarize quotations for each relevant concept, for each participant, in the memo.
5	Visualizing Summarized Findings	Create a table of participants by relevant concepts with summarized findings in each cell.
6	Looking for Relationships in the Data	Reflect on the data in the spreadsheet and look for relationships that address the research question.
7	Abstracting	Consider creating higher-level categories from lower-level concepts.
8	Reflecting	Try to understand how the higher-level category relates to other categories in psychology (e.g., in my study Interactional Identity is seen as linked to action, roles, and personality).
9	Verification	Ensure that the higher-level category found can be seen in every case (e.g., in my study every participant’s data reflects an Interactional Identity).
10	Axial Coding	Look carefully at the relevant codes and demographic data, and study their relationships. Create a model if possible. (e.g., in this study this step produced a model of Interactional Identity with interrelated concepts, categories and demographic variables).

Appendix F: Nineteen Further Case Studies of Interactional Identities

Below I justify the Interactional Identity designations of 19 participants in this study. Each description follows an identical format. To begin I provide the handle and phrase for the Interactional Identity for a participant. The phrase, constructed with the help of the Free Online Dictionary, expresses the sense of the handle. Two paragraphs then follow. In the first paragraph I have used superscripted numbers, such as this ¹⁾ to highlight and enumerate the main reasons for determining the specific Interactional Identity. These reasons correspond directly to the four high-level concepts of the model, namely the individual's 1) Personal Goals for the Project, 2) their View of Who Determines the User Interface, 3) View of Contradictions/Tensions Generally, and 4) View of Design Artefacts Generally.

The second paragraph looks more specifically at a participant's view of designer-developer interactions and the evidence for the Interactional Identity in these data. I summarize the individual's 5) Views of Contradictions/Tensions in Designer-Developer Interactions and 6) Views of Artefacts in Designer-Developer Interactions. Numbered phrases therefore are the key pieces of evidence for the Interactional Identity.

For each individual I then discuss specific designer-developer tensions perceived by the participants, and specific artefacts that they use in their interactions that help in the resolution of these tensions, to show how the Interactional Identity is tied up with how one specialist interacts with the other through artefacts to resolve contradictions. Quotes allow participants to "speak for themselves" and were chosen to support the case for the Interactional Identity and to show how participants use artefacts in their interaction with their fellow designer (or developer). I do not compare individuals with each other; I simply make the case for their specific Interactional Identity using evidence from

interview transcripts. I described designers and then developers. I present individuals in each group in decreasing order of experience.

To read about a particular Interactional Identity page numbers are given here: elder, 432; artist-in-business, 435; teacher, 437 ; facilitator, 440; steward, 442; quality advocate, 444 ; reformer, 447; closer, 449; negotiator, 451 ; pioneer, 453; incorporator, 456; pragmatist, 458; training advocate, 460; historical anomaly, 463 ; practitioner, 466; problem solver, 468; catalyst, 470 ; assembler/builder, 472.

Designers

Interactional identity of Elder, more than 40 years experience. Alan filled a leadership role in his organization. His professed role was IT systems analyst/consultant/point of sales expert. He was the lead designer for not only the software, but also the hardware of a point of sales system. He was the most mature designer in this study in terms of the number of years of his designer-related experience.

¹⁾ His personal goal was to ensure that the various specialists on the projects all had the same goal, which he considered to be a new objective historically speaking, because he believed the existence of multiple software specialists on a project was a relatively new development historically speaking. ²⁾ His view of the user interface and who determines it, is that the designers create the designs, confirming them with the end users, and that the technical people turn the design into working code. He understands this relationship historically in the sense that he understands how the role of the developer was once very different in the past, and how the role of the designer has emerged relatively recently. ³⁾ He understands contradictions as evolutionary pressures on the project or evolutionary pressures that no longer have an impact on a project. For example, Alan sees the current general appreciation in the public for what computers can do now as a positive force in

comparison to previous times when the public was less computer literate. Similarly, “technologization” is seen by Alan as a force that has brought the project into existence because end users (clerks) are now savvy enough for the product.⁴⁾ His view of artefacts is that they are tools, practices and processes that are developed over time and that can turn normal people into experts. However, in the specific project that I observed, he admits that the particular artefacts in use were suboptimal because of the particular skill set of the people involved. Essentially, the designers used PowerPoint to draw the interfaces because they did not have the time and programming skills to master the tool to generate the software for the user interface from a graphic specification. In fact, this is what Alan saw as the major contradiction in the project, one that he expects will be resolved with inevitable changes in the education of designers generally, or possibly through the development of better tools. Alan is the person who proposed a new role for this project called the designer/developer. Four designer/developers were skilled with the tool that allowed them to redraw the designers’ screens and generate the code for the user interface.

⁵⁾ Today, Alan says, developers collaborate with designers comfortably and there are relatively few tensions because developers are more “normal” people now in comparison to previous times when developers were a very elite group.⁶⁾ Alan generally sees issues from a broader perspective, and not surprisingly, the key artefact that Alan generates for the project are architectural plans, keyboard layouts, screen flow diagrams, and keyboard mappings. All of these artefacts provide a big picture of the end product that captures his many years of experience—Alan believes tools embody knowledge. These provide the overall vision and help the designers and the designer/developers to organize their activity of designing and coding many screens for the developers who

develop the back end and connect it to the implemented user interface or front end. What characterizes Alan's interactions with other designers and the designer/developers in his organization is his "big-picture" perspective, which is both historical, high-level, and broad in scope.

Here Alan discusses how developers have changed over time. This quote reveals his views on the emergence of specialized experts in the field of software development. The comments were made in the context of his current project that brought together many specialists including analysts, developers, designers and designer/developers:

Alan: OK, and again: what am I designing and where am I designing? So expertise is following—you know, I now have programmers who are good at different platforms, different levels. OK? The guy who can...who wrote the code that makes this work is not the same guy who's writing the code that makes this system out here work.

Interviewer: So you—?

Alan: Totally different skill sets, same title. OK? So it's...it's that. It's: you've gone from having to have very intelligent people [developers] who had to learn everything to: I can now specialize [be a database specialist or a software architect etc.] and become very good at what I do and not have to worry about the other pieces of it getting done.

Below he links artefacts and the accumulation of knowledge in a discipline when he discusses how an abstract artefact, in this case guiding principles or best practices used by the designers on his team, embody the knowledge of previous experts:

Alan: Right, then the PC revolution happened. And now you've got massive computing and massive projects taking place. You had to, basically, get organized and start following documented design—start following documented project principles. And so design is just a piece of those. [...] Now you start developing tools to actually record, and methodologies around those tools, to put these artefacts into perspective.

Interviewer: What do you mean by "putting the artefacts into perspective around the methodologies"?

Alan: So you actually have guiding principles. So you start using guiding principles and terms like, today it's called Best Practices. So, today you have enough history of things that, ..., OK, you've tried a bunch of things over the years, what works and what doesn't work [and these are crystallized in best practices]?

Alan presented himself as a knowledgeable and experienced *elder*, as “someone who sees himself as an older, influential member of a family, tribe or community,” but also someone who has experienced a great deal of change and is wiser for it.

Artist in business, 40 years experience. Roberto’s role was graphical user interface (GUI) designer and designer of strategic marketing concepts. His task was to collaboratively design a product and design the user interface for a Web application being developed by a startup company that comprised only a handful of people, hence each person served multiple roles. ¹⁾ His personal goal was to “make the user experience more productive [for the end user] ... and more profitable for the end company” (Roberto). Roberto made sure I was aware of his impressive talents as an artist, from the very first email I received which had a remarkable piece of Photoshop work, to the interview where I was shown a recent award, sketches, photographs, and eclectic drawings. ²⁾ Roberto saw designing as both a collaborative and a private activity. Although an artist, he felt that the first priority for the user interface was that business requirements of clients are met, the second is that the design can be extended and the product sold to other clients, and the third is that the design must be usable. ³⁾ When asked about tensions on the project he said that his perspective on tensions was to keep his eye firmly fixed on the real business objectives and not to sweat the small stuff (i.e., the only real tension had to do with setting the direction for the product and company and financial survival). ⁴⁾ To this end, he and the other two or three people in the company (including the developer) met as regularly as they could and artefacts were used to test ideas, communicate, raise problems, and explore the “common ground” between them, which in Roberto’s view

was the user experience (especially cognitive, but also aesthetic) *and* the client wants, which is another indication of his dual business/artistic perspective.

⁵⁾ The key contradiction between Roberto and the developer in Roberto's opinion was the gap between the type of work they both do. He viewed the developer as almost a wizard who "dealt with the new alchemy" (Roberto). ⁶⁾ He felt that the working meetings were critical to bridging this gap. At their working meetings, the developer brought the latest version of the product, the CEO brought PowerPoint slides to present both client and end-user issues, and Roberto might bring screen designs—sometimes arranged creatively to expose a problem. According to Roberto's view, as a group, the team explored these artefacts that represented different perspectives and in the process developed a common ground (a shared understanding of user experience and client wants) so that between meetings they could work independently on their respective pieces of the work. In Roberto's view, the meetings therefore set the direction for the work of the group to ensure their survival as a business.

Here Roberto tells me that the common ground between himself, the developer and the business analyst is "the user experience" and "the client's desires":

Interviewer: So how does he [the developer] ever communicate what he has to do to you? Or does he ever need to?

Roberto: We talk about the function and we talk about what it—what the user will experience. And his tool to get that is, that's the common ground. And what's common for all of us [the designer, the developer and the business analyst] is: What is the user experience? What does the client want and then how does that affect the user experience?

Interviewer: Ok. And then everybody translates back to—

Roberto: That's right. And for [the developer], he wants to make the user experience as fast as possible. So for him the code has to be very efficient and it has to get right to the root of the problem so it displays my graphic as quickly as possible. I have to design my graphics in such a way that they break into small chunks, that noone's waiting for the Mona Lisa to download in order to get [the developer's] numbers crunched into a graphical user display. So it's a balance, it's always a balance, so. But he

understands how the page is gonna be split up and everything else. I basically put down my wish list, I put down the drawing of how I think it should look. And then he goes away [to work on the drawing]. I don't do the drawing on the final show [meaning the final product]—he actually goes and cuts it and makes it, and makes it look just like my drawing.

Roberto lets me know that he's not only a graphic designer, but also a songwriter, cartoonist, and illustrator. Regardless of his creative outlet, whatever artistic media, it is his little sketchbook that quickly and effectively captures important concepts he comes up with:

So then I made my quick little list and I made my quick little notes. And, yeah, so these are some of the notes that I've scribbled down from a [design] meeting [...] These are additional things on: How do I deal with, um, buttons and [other interactive elements in the user interface] and how do I stack by type. So basically it's little thumbnail sketches, which give me an idea of whether it's gonna work or not. [...] Roberto flips through his sketchbook] those are the lyrics for a song I just wrote them. Um, yeah, that's another [word indistinguishable]. Yeah they're all blended together: cartoon strips, fake ads [...] So this is how I write down concepts. And so they're thumbnails, they're little things—this one is a fake ad for a new green product, [...] I thought'd be great. So here, for example, I do the general sketch of the thing I'm doing and then I do a little enlargement so I can capture the feeling of it. [...] I still haven't found anything that I am more comfortable with, [...] for me, it's a pencil and a piece of paper. So, it's a small book, I carry it with me at all times. (Roberto)

Roberto presented himself as someone who was an artist-in-business because he used his artistic skills to achieve business goals for the benefit of the organization.

Teacher, 33 years experience. Massie, a learning design specialist, was charged with designing Web-based learning environments within her organization and being the project manager for the product she was developing. Essentially her task was to determine the structure for a Web course, work out the content in collaboration with her clients, and ensure it was developed and delivered. She was resourceful, responsible, and adaptable. ¹⁾ Her personal goal was to produce a product that was really useful to her end users in the sense that it supported them to learn. ²⁾ Massie feels responsible for the

design of the user interface, but she is a new employee in the organization, and she is very open to learning from others—including the more-experienced developer and her organization’s auditors.³⁾ She tries to resolve contradictions or tensions in the project by identifying something that is “possible to achieve” and that is “meaningful” to those involved, a mindset that one would expect of a teacher.⁴⁾ She uses many artefacts in her work and calls them “props,” “materials,” “supports,” “tools,” “kits,” and “aids,” which are words one might expect a teacher to use. The artefacts she used frequently educated others (including the developer) about the process to be followed, but they were also useful for determining course structure and content. Massie relied on the artefacts heavily to control the project work, and to learn how to design courses herself, because she was relatively inexperienced with respect to the complex processes adopted in the organization.

The key contradiction of the project, from Massie’s perspective, was designing something that really supports end-user learning for clients who do not understand issues related to learning or Web design.⁵⁾ The key contradiction therefore was between the design activity and the client activity and this was experienced jointly by the designer and the developer.⁶⁾ To resolve this contradiction, the designer, who is receiving some pressure from the developer who wishes to become more involved in the project “up front,” uses the developer in way not specified by his role. By recognizing his expertise as a person who creates training material and a person who has experience with creating content for the Web, she uses the developer to help to educate her clients. She does this by organizing meetings, a key designer-developer artefact. For the project that I observed, the developer was atypically involved in the process from the start, invited to formal meetings, and asked to lead meetings (about creating training materials and about

the limitations of the Web as a medium for course content). She also uses the Web course itself, which had been created by the developer's co-op students, as a valuable aid for communicating with the client once the work is in progress. Massie's Interactional Identity as a teacher has allowed her to see the skills that the developer has in the area of training (which one would not expect of a typical developer) and to leverage these talents and her talents as a Web designer to educate her client. She does this skilfully without relinquishing control of her project.

In the following fragment of my interview with Massie, she discusses her relationship with the developer she works with, who is her senior and who is interested in designing courses and not just developing them. She enacts how she would interact with him, which involves using the Socratic method common to teachers. It's a delicate situation. While she welcomes the developer's input and values his experience, which is significantly greater than her own, she would stop short of allowing any developer to take over the design work. Like a classroom teacher, she remains in control.

Interviewer: Do you ever find that there's a grey area where your roles overlap and he [the developer] might do some of what you do and you might do some of what he does, traditionally.

Massie: Yeah, because he's [the developer's] really interested in design too. And you can't not—. You know, like, I mean, some people in design are really interested in development too [...]

Interviewer: Is the developer's interest in design OK with you?

Massie: Oh, yeah, it's OK with me. Yeah. I mean, I suppose it could happen that you'd have to tell somebody [meaning a developer] to back off if they started ah, if they started taking over. But like, they [the developers] don't actually have this upfront piece with learners [normally they are not involved in the initial stages of course design which involves working with the clients], but I invite him [this particular developer] to participate because he's interested. So, like, I mean, if you did have a developer who was saying, "No, I really think you should go in this direction with the course," [and who was at odds with what was going on in designer-client meetings] then ah—then, you know, you'd have to have a meeting [to resolve this tension]. But ah you— when somebody is that interested, [...] you'd probably say where we're trying to go [with the design], you know,

like, and from discussions with the client [you'd say to the developer] "[We know] we wanna go here—how can you help us get there? How?" And "You're interested in learning or you've studied a lot yourself. If you were that particular learner, what would you think?"

Massie's Interactional Identity is *teacher* and not surprisingly, her highest degree was in education. She engages in the courseware production activity as a teacher.

Facilitator, 13 years experience. Frank is a user experience designer and charged with developing screens for a feature being developed for a large graphical user interface application. He worked with developers, managers, and fellow designers on a daily basis and saw himself as being in the middle of the activity.¹⁾ His personal goal is that end users benefit through his design work. One of the ways he does this is that he learns about and develops tools for design work, which is one of his passions, and which was a way that as a designer he was unique.²⁾ He believes the designer owns the user interface design, but the designer brings a lot together (i.e., he ties a lot of feedback together when he designs). The designer considers the viewpoints of the user, the engineers (a.k.a. developers) and business people. Essentially, Frank sees the user interface design as the "problem" that he must solve.³⁾ He sees contradictions as part of the occupation of being a designer and that these are resolved by talking to people, creating and presenting the right artefacts and developing one's toolkit.⁴⁾ In Frank's view, artefacts facilitate group problem-solving. Artefacts are carefully selected and crafted for presentations depending on what he wants to convey and he, more than anyone, had a huge range of artefacts types in his tool kit. Some artefacts (like standards) embody previous solutions to problems.

⁵⁾ Frank experienced no tensions with the developer with whom he worked.⁶⁾ Their shared artefacts directly focused on the work of designing the user interface and

included product proxies of many types including screen drawings, flow diagrams that depicted interactions, and specifications. The developer responded to these artefacts promptly and in great detail, much to Frank's satisfaction, and she had a significant impact on the design. From Frank's perspective, she was an excellent participant in the distributed problem solving-activity in which he was engaged. He also appreciated that the developer went to great lengths to make her software available to him. She had to innovate and work hard to do this. This helped Frank keep in touch with what was actually being developed from his design specs and helped him to respond to any problems that arose during development. In the project Frank and I discussed, significant issues did arise related to the globalization of the product.

Here Frank talks about how as a designer he “goes out” to deal with the “business side” and the “engineering side” on behalf of “end users” to come to a design solution.

Frank: So we work for visual design-visual interaction designers, the UI architects, ah, to define, um, a solution. And we're also working closely with Product Management and Engineering, to understand the technical as well as the business ah problems that we need to solve.

Interviewer: OK.

Frank: So basically though that's the key three. So there's PMs for—or there's basically PMs to represent business, engineers to represent the technical, and then to UI design to represent the users—and we form some kind of like a triangle. [...] So basically the interaction designers, and maybe the [UI] architects too, are the main people who kind of go out and actually deal with other groups outside of the UI team.

With regard to the developer specifically, Frank describes how he elicits very early feedback from her on the design:

So, yeah, [the developer's] role would be [...] exploratory initially – [...] what can we actually do. [...] [The developer] can also help me to understand: how big a solution I'm proposing might be, how difficult it will be, or if it matches anything else that we have out [in the market] and that's also in the product already—things like that. So that I can also refine my designs that way with [...] an understanding of how the product actually works underneath.

Frank presented himself as a *facilitator* of distributed problem solvers. As a facilitator he is someone who “makes progress easier, and contributes to the fulfillment of a need or furtherance of an effort or purpose.”

Steward, 12 years experience. Mason is a business and interface designer in his organization where he creates Web products. He sees himself as an interaction designer. He is involved in the project from concept design to end-user testing through field trials and focus groups. Though not the person in charge of the project, he watches over every detail of the project as it relates to design issues and his main orientation is to move the design along, and specifically to move it in a direction where the end user will be satisfied with the outcome. ¹⁾ His personal goal is to rescue the project (he perceives that the client has made some very poor choices) and ensure that the design meets a host of objectives including that it is “implementable,” accessible, meets Web standards, has the right functionality, is responsive, and intuitive. ²⁾ In terms of the ownership of the design he sees himself as an egoless designer, and as such he is very open to suggestions from others and in fact relies on these suggestions to design. ³⁾ He believes that contradictions can be solved by heroic individuals, and by people getting together, talking openly and using common sense, thus he likes meetings and enjoys presenting. ⁴⁾ Artefacts, in Mason’s view, help people to take charge of a situation and to put forward a point of view, but they also help to progress the work in many ways such as helping people to get their heads around problems, to illustrate something, to help people see the big picture, to support the process of checking and balancing work, to help people determine the scope of a problem and so forth.

⁵⁾ The major project conflict in Mason’s project was between the designer’s work and the client. Mason, in general, had a very good relationship with the developer, but he

says that developers, including the developer he was working with, are people who want to complete as fast as they can. They build the product by processing tickets (work orders), but they take a short-term view (i.e., they do not consistently see things from the end user's perspective as he does and as a steward would). He also sees developers as professionals who have good ideas about the design and who do a lot of work.⁶⁾ Mason was able to resolve the contradiction between the long-term perspective of himself as designer and the perceived short-term perspective of the developer by effectively using ticket artefacts. These allow him to specify manageable chunks of work for the developer. Effectively, the ticket is the primary tool that Mason uses as a steward, to guide the developer along so that Mason's long-term goals are met. The most useful aspect of the ticket is the product proxy, which the designer created. This was an HTML interactive, high-fidelity version of the product to be created that had Web and accessibility standards built into it. These are used without modification by the developer. In this way the designer ensures, like a responsible steward, that essential standards are met and end-user experience is therefore adequate in these regards.

Below Mason tells me about his role. He talks about how as a steward he would feel obliged to fight really hard with future clients on behalf of his user community so as not to have parts of the design outsourced in the future [a decision that has been a major headache for him on the project I followed].

Mason: But at the end of the day we are the stewards, right? So we do have to, you know, decide when to cross the line.

Interviewer: OK.

Mason: We rarely ever do that. It's only with [name withheld] that we've ever done it. The two times we've done it are with her. And if we ever did do it again [outsource some of the design work], I think I'd fight it really hard. [Having "the look" outsourced, really constrained what the designer could do in terms of the interaction.]

Interviewer: Do what again?

Mason: Outsource the design work.

Mason organizes the developer's work through a ticketing system (work orders) that is organized around user stories. In this segment he describes the work to be done and how he elicits feedback from the developers and other team members by reviewing tickets and a mock-up.

So from then, they [the developers] get tickets [organized around a single user story] assigned to them. Um, you know, I may review the tickets with them, to say, 'Hey, do you see anything that we've missed?' do you know what I mean? So ideally we'd have [...] a meeting and go through the whole Web site for them, to give them a feel for the whole thing. Looking through the mock-ups. And then, when they actually get a ticket, we would review that again with them before they do the ticket. (Mason)

Mason is a *steward*, someone who "manages another's property." He acts as the team's steward for producing useful software for his product's target audience.

Quality advocate, 11 years experience. Nick is a senior user interaction designer and usually the lead designer for graphic user interface and increasingly Web-based applications in this organization.¹⁾ His personal goal on the project was to develop the HTML code for the user interface, determine the look and feel, and ensure aesthetic quality.²⁾ When in meetings and presenting his designs for the user interface he says he checks his ego at the door and tries very hard to listen to everyone's opinion, including the developers.³⁾ With regard to contradictions he says that sometimes there's just too much compromising, especially with quality and that all parties should be involved in resolving contradictions. In his view, the compromises are mostly imposed by the business people in his organization and are equally unsatisfactory to both him and the developers.⁴⁾ He views artefacts as objects that support a negotiation process, to determine what will be developed to what standards and when. They can also be used to

communicate a position and also to organize the work, once it is determined what will be implemented and what will not.

Nick described a very satisfying project where he had excellent rapport with the developer; ⁵⁾ however, in general he told me a story of how quality, which he saw as the concern of the designers, was always compromised in the company. ⁶⁾ In other projects he worked on where developers were “hostile,” Nick developed the strategy of writing design specifications that were essentially very long developer to-do lists that would ensure quality products were built. These were essentially very detailed, textual, numbered requirements (most requirements had to do with the user interface, but some were financial and contributed by the financial analysts). He says that usually the numbered requirements get cut in meetings with the financial analysts and developers in favour of meeting deadlines or addressing developer issues. However, the developer with whom he had excellent rapport loved documents in the form of to-do lists because it made it easy for the developer to convey estimates of the time to complete work and it made it easy for the developer to organize his work. Another key artefact Nick created, the wireframe, helped Nick to convey what he would like to see built, essentially the quality product that he envisions. Nick is always an advocate for high standards, which by this he means good design; however, he has no power over the developers or the analysts. In his organization, the financial analysts have power over both the designers and the developers. Nick presents himself as someone who is battling to do his best to ensure products are a high quality, but also as someone who has lost many battles and who expects to lose more. While he had nothing but admiration and respect for the particular developer on the project Nick and I discussed, Nick, returned repeatedly to the

larger story of conflict within his organization so this dominated my impression of his Interactional Identity.

Here Nick talks about the importance of quality.

Nick: Have you ever heard of the iron triangle of software development? [...] It's ah—it's money, ah, features and time and quality in the middle. And one of those always has to give. So: you want more features? Well, you've got to either put in more resources—so this is resources, cost, etc. Time is schedule, and features are just, you know, functionality and stuff. So you want more functionality? Well, you either hire more resources or you let the time slip. And every group has their own, um—what would you call it?—what-they-would-prefer. So, you know, the directors and the managers have promised time. Um, developers care about quality and functionality and, ah, sometimes cost. Ah, management is very concerned with cost so they don't wanna add more to the project. But it's always a push/pull, and what ends up suffering all the time is [what the designer cares about which is] quality. [When interacting with the developers over proposed designs] you start compromising the design. Which is inevitable, it's going to happen. But so often the compromise is so far back that it's—you have to ask, is it even worth either developing it or doing it because you're compromising so much that... So, that happens a lot. So—

Interviewer: How do you work that out, you know, these compromises or these tradeoffs between what functions will appear?

Nick: You have to let, ah, the powers-that-be make that decision: So here's the design that we think, or we've tested, or whatever it is, um, that you're OK with; it's going to take X amount of time to develop; if you compromise it this much, it's gonna take Y amount of time to develop. And more often than not, they compromise quality in order to meet deadlines and have time to do other pieces of functionality. [...] In my experience they'd [the “powers that be” would] rather deliver 3 to 7 pieces of functionality and compromise on UI because—we'll fix it later. Never happens. The end users, ah—the care for the end users is very minimal. Because, it's a million dollar software, the end users are not thousands of users, they're 5 users, um, and they'll learn. They'll learn through, um, trial and error, they'll learn through tutorials, they'll learn through, ah, you know, [...] a tutor [...]. So we'll [the organization will] compromise on [the] UI in order to get features and meet deadlines. So it happens quite a bit.

Nick presented himself as a *quality advocate* (i.e., “one who argues for quality; a supporter or defender of quality standards; one that pleads in another's behalf for quality; an intercessor between parties who advocates for quality”). When he interacts he is a strong advocate for quality in software products.

Reformer, 10 years experience. Muriel is an interaction design team lead and leader of a standards group in her organization. ¹⁾ She managed all the designers, and had very little involvement in the project I observed, other than providing a designer for it. ²⁾ For all projects she believed that designers were responsible for their designs and especially to adhering to Web and accessibility standards. ³⁾ She saw problems and hence contradictions “everywhere” and saw plenty of room for improvement. ⁴⁾ Artefacts might be about work that needed to be done, or issues to be resolved.

⁵⁾ She believes the main issue/tension in her workplace is a lack of adherence to a process (especially among the developers) and an inability on the designers’ part to regularly enforce standards. She believes the designers are responsible for the design and that includes adhering to the Web and accessibility standards of her organization (which her designers do not always ensure). She sees many problems and much room for improvement. ⁶⁾ She believes artefacts are there to manage the work. Some of this management occurs on a daily basis; for instance, some artefacts represent what needs to be done, like tickets and stories. Others are points of discussion such as site maps. Others such as the project wiki, and release and information plans record all the information about a project and these she calls “management-side artefacts” because they help her to reach business goals. Other artefacts Muriel uses ensure reforms. On some of her projects, interactive, realistic mock-ups that incorporate necessary Web and accessibility standards force compliance on the developers. A number of her designers’ projects do not adhere to standards because some of her designers cannot produce the type of artefact that ensures Web and accessibility compliance. To get around this, she hires designers with Web-development skills. She also wants to transform the workplace by introducing the role of a manager of developers so that she does not have to address standards issues

on a project-by-project basis. She uses meetings to argue for this new position in the organization. She also uses meetings to educate people about the tensions she experiences in the workplace so that they understand that the lack of adherence to standards is systemic.

Below Muriel describes her efforts to find a champion for a process she believes the designers and developers should consistently follow:

Before I left [Muriel had been away for a year and a half], things were working very well. We had a , [...] a person who like evangelizes the process, who leads it. [...] We have someone that basically takes it on and says, “This is what we’re gonna do.” [...] And that person gets real excited about it. And everyone else gets excited about it. Well, that person left. So now I find we’re in this really funny place of me being the one saying, “We’ve defined a process. Why aren’t we using the process? Are some people using the process? Are other people not?” So personally what I see lacking is that there’s that developer team-lead [role] that is missing—[the person] to tell people [the designers and developers] what they’re supposed to be doing in terms of the development [process], but I guess [now] it really falls upon the PMs [product managers]. [...] [But,] it’s really hard because you have each PM doing their own thing. And the PMs are really the ones that have to drive the process, right? They’re the ones that are telling their team, “This is the way we’re gonna do it, this is the way we’re gonna run with our project.” If they’re not telling them how to do it [properly], then it’s not gonna happen in that way. So it doesn’t matter what I tell my people [her designers]—it’s whatever their PM tells them. So really I don’t have any control in that. So I find that really frustrating. ‘Cause people look to me to help with the process—‘cause I’ve been here for a long time now [...] but I can’t do it in my own little bubble, you know? [...] I don’t have control over the PMs. (Muriel)

In her interactions with the developers she advises her designers to use a particular approach to creating mock-ups that leaves the designers in control of more of the user interface.

For me it’s about efficiency, right? And that’s why I’m trying to convince my team to get into MyEclipse and to be able to go into the code. ‘Cause, for me to tell a developer sometimes, “You need to take this one thing here and move it one pixel left.” and they look at me like I’m nuts. And they’d look at it and they’d go, “Well, I don’t write HTML, I have no clue how to do that, I can’t get that button to move.” Or, you know, especially now that we’re in CSS it’s even worse because they don’t know how to do the CSS part of it. And they don’t really want to do the CSS part of it. ‘Cause it’s not their job, it’s our job. So instead of us

having this 10 minute discussion about how to get this thing to move one pixel over, or me sitting at their desk to get it to move one pixel over, I'll just go in and do it [...] I just find it more efficient than trying to discuss it; I just gotta do it myself. I don't know if that's good or bad, but ... (Muriel)

In her interview Muriel presented herself as a *reformer* who wanted to change the organizational structure at her workplace to benefit all projects. The transformed organization would achieve greater compliance to design standards.

Closer, 8 years experience. Sean is a user interface designer in his organization and was charged with designing all of the screens for a Web product under redevelopment. ¹⁾ His personal goal was getting all parties to agree to a single design and he worked to create a design that management and R&D would “buy into.” His screen design work was critical to achieving this goal. ²⁾ Sean owns the design of the user interface, but he tries to reflect what the managers want, what the developers need, and what he believes is good design. ³⁾ Sean therefore sees his key artefact, his screen designs, as “a deal” that he puts on the table in meetings where many stakeholders are present including the developer, the product line manager, documentation staff, quality assurance staff, and the managers of some of these people. In Sean's view, artefacts support specific tasks that need to be performed to settle issues and reach agreements. ⁴⁾ He sees contradictions as tensions between individuals on the project that pull the project in different directions, whereas he wants just one direction. When he presents in meetings he genuinely wants input from all parties to craft a new deal (a new set of screen designs) that is more acceptable. Like many of the designers, he viewed himself as someone whose unique skill was that he could take many aspects of the project into consideration at once and balance one against the other.

⁵⁾ Sean viewed the developer on the project as someone who wanted to develop a modern and maintainable application, but also as someone who was not particularly aware of the needs of the product line manager and other project stakeholders. The designer wants to develop an application that satisfies many criteria: the PLM's (Product Line Manager's), the developers', and other stakeholders' and this was the major contradiction between Sean and the developer, according to Sean. ⁶⁾ Working with the developer involved organizing meetings where the views of all the other stakeholders were exposed. Sean forced the resolution of conflicting views (principally between the developer and the PLM) by presenting realistic drawings that depicted the future product in formal meetings thereby forcing a concrete discussion of the proposed solution. The developers tended to speak in abstract terms and the PLM, although he knew what he didn't like, wasn't sure what he wanted.

Here Sean describes how his role is to find "one direction," in essence to mediate between the business people and the developers.

So the tension is, me trying to force them to come up with one direction. Right? So in this case the PLMs [product line managers] are asking for one thing and the R&D team [i.e., the developer team] has something fairly different in mind. So you call a meeting with both those sides together and you get them talking to each other. And then they will see: well, the R&D guys [i.e., the developers] are saying, um, that we want this and this and the PLM [product line manager] is actually saying No, I don't want that. And then the R&D guys will say, Well, we have to do it slightly differently because the—you know, the thin client isn't gonna behave the same way as a thick client and you're gonna have to make some compromises. And then they'll start talking about the same thing, right? Other than talking about things differently. (Sean)

This quote from Sean reflects how artefacts are a focal point for discussion in this process.

You would have something in your hands. Usually. Unless it's something that's more of a work flow—well, even then, you know, (I would) create a flow diagram or some sort of drawing—right?—and say, "Is this what you thought?" Right?

And I think that's what you're seeing this week—right?—or this past week is, ah, in this sort of initial environment, it's like, me trying to figure out what these people are actually saying? Right? 'Cause there's a lot of people saying things. And trying to get people to talk on the same page, right? And me trying to take that information then do something with it that makes sense. (Sean)

Sean presents himself as someone who is a *closer on a deal* because he sees his design work as “closing in on a deal” that is agreeable to all stakeholders, including himself.

Negotiator, six years experience. Jay is a user interface designer in her organization. Her task is to develop approximately half of the screens for a complex point of sale system that is being completely redesigned. She is generally very busy and is working on approximately half a dozen features at once. She interacts with the lead interaction designer, the analysts, the designer/developers and the developers on a daily basis. She sees the project goals as being the individual goals of the organization's managers (a straight port), the business analysts (work flow changes and more maintainable software), and the lead interaction designer (a new interface style). ¹⁾ Her personal goals are to design the screens in the new look and feel, incorporating business changes into the workflow, while retaining all other aspects of the old system (the analysts have, much to her frustration, largely ignored the latter aspect). ²⁾ She believes that as a designer, she owns the user interface, but that she must please a host of others. ³⁾ She sees contradictions as tensions to be resolved by compromises and that design is about negotiations. ⁴⁾ She makes heavy use of artefacts because, for her, they expose what someone wants or thinks, they are aids to getting feedback, they help people to reach agreements, they help to move the work forward, and they help to support clear thinking, all of which are essential actions when negotiating.

⁵⁾ Jay sees the designer/developers as the people who faithfully reproduce her designs and she has a very good, tension-free relationship with them. ⁶⁾ The key artefacts she uses to communicate with project team members are PowerPoint slides of screen flows and detailed screen layouts. These are central to the work of the project. Designers, designer/developers and back-end developers regularly carry paper versions of Jay's slides from desk to desk for quick chats. The only ongoing contradiction between the designer and developer Jay experiences is that the designer/developer faithfully reproduces the designer's drawings so that his work is pixel perfect relative to her drawings, even when this is not a requirement. Because there are no apparent negative consequences of reproducing her designs in a pixel-perfect manner, except that it may increase the designer/developer's workload, she does not try to resolve this issue. Because the designer/developer accepts these drawings as largely 'nonnegotiable', there is very little negotiating required in this particular relationship so her Interactional Identity does not emerge strongly in her relationship with the developer. The exception would be when he offers his opinion on her work when he is first presented with one of her designs. At this point, because she values his opinion, some negotiating over the design may occur.

Here Jay tells me about her work, which is laden with tension, but not designer-designer/developer tension:

Jay: Oh, it's all about tension, it's all about negotiation.

Interviewer: OK. Good, please tell me more.

Jay: Well, like I said, we have to manage the conflicts between what the business analysts think they're getting and what the developers can actually deliver. And, on the third side, from what the clerks, or the customers, can actually bring to the table ... [and] how a user actually is going to use this system—their expectations, their previous learning from their previous system. Um, and how people will use a touch screen and a keyboard to perform their task.

Below she provides an example of her negotiating work on behalf of the end user (the clerk) where she must resolve the conflicting priorities of the business analyst and the back-end developers:

Jay: It's a juggling act. It's a...it's a task of compromise; it's a task of rationalizing priorities, conflicting priorities sometimes.

Interviewer: Could you talk a bit about those?

Jay: Let me think. You probably want—it's best to probably give you an example of that. One of the screens has a new business requirement, for example, to require the [...] clerk to check a [...] code before they proceed with [...] another [...] form. The current process is manual; they tell the customer to go away and check it in the book [later]. The new business requirement is to bring that checking facility online and give it to the clerk.

Interviewer: Ah, OK.

Jay: So that's a new business requirement, which doesn't exist in the old software. The problem is it's not so easy to implement. The business analysts assume that this can be done easily. Um, technically it's a reasonably straightforward thing for the software people to do. But as far as the clerk's perspective, it's difficult to do. Mainly because of, um, the hardware. They need, in their work flow, to build in time to dial out, go to another Web site, come back, bring in the [bit of] information, process it, and manage it with all the [other] information on the screen that [...] they're managing at the same time. It's not a straightforward task. So the job of the interface designer is to bridge the two: the business requirement, which says you must check the [...] code before you proceed with the rest of this information, 'cause we don't want bad information getting in the system, and the developer's ability to actually implement that. In the middle sits the poor clerk [whom the designer represents] who has to manage it somehow on a screen, which is not straightforward.

Jay is a *negotiator*, or “one who confers with another or others to come to terms or reach an agreement.” In her design work she negotiates or confers with others to make design decisions.

Pioneer, 2 years experience. Kate is a user interaction designer in her organization. She was the sole designer involved in the redesign of a graphical user interface product. ¹⁾ Her personal goal was to create a product that fit the work practices of her end users over many tasks and over long periods of time. ²⁾ Her view of the user interface is that it is the designer's responsibility, but that it can't be done properly

without access to clients and end users.³⁾ She believes tensions in the project should be resolved directly and not avoided.⁴⁾ Artefacts generally helped her to develop her ideas. For example, she used prototypes to help her to run through scenarios. Client meetings, which are very hard to organize because of barriers within the business side of the organization, and which she uses very creative strategies to overcome, help her to uncover user needs, and explain concepts, which she does by telling stories. Much of Kate's work is as a user needs analyst, which made her unique among the designers. Kate's other task is to produce sketches of the user interface for the designers, which she is encouraged to do collaboratively.

Meetings with the developers, where Kate and other people on the team present their concerns, are generally very unsatisfactory experiences for Kate. She described developers as people who do whatever they want and whose priority is not the user interface. Kate had huge difficulties engaging the developers in caring for her designs or adhering to her designs.⁵⁾ The major tension between the designer and developer from her perspective was how to be an effective designer within an organization that values business and development over design and that does not give power to the designers.⁶⁾ For Kate, artefacts she creates communicate what she understands the user wants. She develops wire frames, prototypes and tells stories during presentations of this material. Artefacts can also ensure the work on the user interface is completed by the developers. She enters user interface issues into a project spreadsheet and has them discussed at meetings, which was a solution introduced by a meeting facilitator, which Kate adopted with gusto. Once in sheer frustration with the developer group, her design group hired a user interface developer themselves and had him join the group of developers, but just to

implement the changes she wanted. In her sheer tenacity and refusal to accept the status quo in her organization, Kate saw herself as a pioneer.

Here Kate talks about how she found a unique solution to the problem she experienced with the developers not making changes to the user interface that she requested and the managers not supporting designers.

Katie: It was when we were having a lot of pain getting them [the developers] to fix UI issues—because the developers were doing the UI [user interface], but they were also doing all the back-end stuff, too. So from a development perspective, what's more important is that it's [the server's] actually working, and working the way it's supposed to be, rather than how it looks. Whereas us [from a designer's perspective], if it's not—if it doesn't look and work on the end that the user's interacting with [the Web client], that's as big of a problem than if it's not working right on the back end. But because they [the developers] had a different perspective, in their mind, like, it [anything to do with the Web client] was all low priority. And that's how it gets spun to the business line as well. And to the business line, you can understand, it seems reasonable. If it's not working right, what's the point in making sure the UI looks right if in the end it's not gonna spit out the right stuff. So they'd be like, OK, I can see how that's a priority. So in the end, our hands were kind of tied because we couldn't get what was a priority for us fixed. Which is why our prototyper got sent down to New York was to try and help out on fixing the UI. [...] And when he left, that's why we hired a contractor was because we wanted the UI fixed. So he came in to, like, work with development remotely. [...] He would work with the developers on like: OK, I'm gonna work on this piece, and he would fix ... he would go ... he would work really closely with me, fixing the UI. And then even some back-end problems that they had, he would help them out when all of our UI issues had been resolved.

Interviewer: Wow.

Katie: Yeah basically we hired our own UI software developer to come in, who I worked with, and that was that.

Kate presented herself as a *pioneer*, “someone who ventures into unknown or unclaimed territory to settle; or someone who opens up new areas of thought, research, or development.” She particularly wanted to transform the workplace to better meet user needs.

Incorporator, 2 years experience. Alice is the learning and content designer in her organization and she is charged with ensuring learning and content are present in gaming products. She works with several other designers. ¹⁾ She wants her work to align with the work of others so that a cohesive product ensues. ²⁾ She says a successful user interface design is one where the work of all the designers is successfully integrated, which she says is a fun process that involves interacting with her fellow designers and the developer, who also acts as a designer of interactional gaming elements. ³⁾ She believes that her company sets up contradictions deliberately because they help with the game development. She believes new designers (such as a recently hired usability specialist) add new tensions, but ultimately their presence results in a better game as these tensions are resolved (i.e., all the designers' work is integrated or incorporated into a whole product). ⁴⁾ Artefacts, she says, exist to support communication (i.e., they show "what a person wants" or in the case of the current version of the game, "what exists"). She responds to artefacts by assessing their implications for her content and learning designs. Her preferred artefact is the current version of the game itself because it shows everybody's work in relation to everybody else's work.

The developer is key to allowing Alice to incorporate her work, in that he continuously recreates the game so that she can check how well her work fits relative to the work of other designers. ⁵⁾ Ideally, Alice would like to be able to create the game herself, but she must rely on the developer to recreate the game, which is the only ongoing tension with the developer that she expressed in our interview. ⁶⁾ Alice's preferred mediated method of interacting with the developer is talking and she values working meetings and chats highly. Alice suggested significant changes to the game in working meetings after seeing the latest version of the game. She wanted to ensure

learning occurred in “manageable chunks,” a principle she wanted incorporated into the game and her concrete suggestions that embodied this principle were built into the product by the developer. As content designer, she saw her task as incorporating her own ideas and the ideas that others had about content into the product. She made numerous false starts producing the content and she engaged in many one-on-one chats with the lead designer and developer to get their ideas on how to organize and produce the content. With the lead game designer, it was how much content there would be. With the developer her goal was to produce the content in a format that he could automatically transform into the game. Some of the talk about content with the lead designer and developer was crystallized in the form of sketches, annotated tables, or calculations that estimated the amount of work for a particular organization of the content. Alice used these artefacts to completely reorganize her very early work on content production. She also kept in constant touch with the developer to align his work and hers because there were many minor daily misunderstandings about the game. She would forward significant segments of the lead designer’s documents to the developer knowing he did not read documents, but he did read email, so that the three of them were aligned on the direction and details of the game. This work is indicative of someone whose aim was to have her work fall in line with the work of others, to make the game that the lead designer envisioned, which was Alice’s professed project goal. She saw artefacts as aids to alignment work.

Below Alice describes her role in the team and the developer’s dual developer/designer role, emphasizing collaboration and working with others.

My role in this particular project is to work with the lead game designer who is also the CTO in that he is the one who drives the development of an R & D project, so in this case it would be [name withheld], so [the developer] and I, [the

developer] is [also] a game designer for R & D projects taking [the lead game designer's] leadership, just like I'm a content designer for an R & D project, taking [the lead game designer's] leadership. So, my role and the whole team's role is to really collaborate with [name withheld] in driving the vision of a new R & D project. (Alice)

Below she describes the developer's responsibility with respect to her work (designing content and ensuring the game supports learning) and the work of others including the lead game designer's, the graphic designers, the usability specialist's.

Alice: So the responsibility of the software developer is that he can translate [the lead game designer's] design vision into game functionality.

Interviewer: And In relation to the content of the game what is his responsibility?

Alice: In relation to the content of the game he is very integral in working together with me to make sure that the content and, and other designers [contributions] such as [name withheld] as the usability designer and [name withheld] as the graphic's designer to make sure that the content is displayed in an optimal way from both a playability and usability perspective. So often times [the developer] will have, for example, his idea was [...] to have the little box where the face comes in of the player [this is like speech balloons in comics]—the little action description is defined on [the] top [part of the box] and then you have your list of options [in the other part of the box]. [...] So [the developer] really has freedom to interpret what's asked of him in the best way possible and then we work together [...] and I might have an opinion and he might have an opinion and then we chose one [that works best overall] ...

Interviewer: So he's not just the kind of person who builds what someone else tells him to build.

Alice: Not entirely. He's definitely a designer as well.

Alice presents herself as an *incorporator* in her interview, someone who “unites one thing with something else already in existence.” At work she sees her task as incorporating her design work with the work of others.

Developers

Pragmatist, 20 years experience. Jack is a founding developer and a software architect on a Web development project in his organization. He says that “perfect (the unachievable ideal) is the enemy of good enough (the more achievable ideal)” (Jack).

Interviewer: What really helps you see a simplification? Or identify one?

Jack: Again, the working system. ...—I've always worked—oh boy, even before I was building software professionally, probably even in high school—worked from the, you know, make-it-work-make-it-pretty principle. Make it provide some value and then sort of iterate on whether it's—ah, how well or how easily it's providing that value. And sometimes you'll find that, it'll get to a point where the value that it's providing, while still simple, is much more than what it would cost to get it to that extra level where it would be even simpler or better or more elegant or whatever--it's just not worth that extra cost or time. You know, time and cost can be considered, ah, the same. I heard just recently a great statement saying that 'perfect is the enemy of good enough.' And finding out what's "good enough" is the key.

Jack is achievement focused. ¹⁾ His personal goal for the project is to leverage the creativity of others who determine the look, feel and functionality (the higher ideal) to produce an application that performs well and provides the required functionality (the achievable ideal). ²⁾ He believes the user interface is mainly the responsibility of the designer and the president of his organization. The president has an ear to the marketplace, clients, and end users while the designer suggests concepts, ideas and is generally creative. ³⁾ He sees conflict or tensions as a good thing because it pushes the envelope and produces a better product; however, he sees that the limiting resource of the project is the amount of coding he can do in a given time. He says the main contradiction is balancing value, what the designer and the president envision as a product, and cost, what he can code in the time he has committed to the project. He believes his time must always be considered when suggesting a simplification, improvement, or increasing the elegance of the user interface. ⁴⁾ His view of artefacts is that they are a means to an end and he sees them as cognitive aids for individual or group problem-solving.

⁵⁾ In interacting with the designer, Jack reports only very minor tensions. Sometimes the designer does not know the real constraints of development with regard to what is feasible, but this is also true for the business representative, in this case the

president of the organization. Also, sometimes the designer does not consider the costs, in terms of his time of proposed user interface concepts.⁶⁾ The key artefact Jack shares is the current real version of the product, which is what really exists at any given point in time. This is what he brings to every meeting. Other artefacts are “necessary waste.”

When you’re building software, the working system is the primary artefact. Everything else serves to get to that goal. If you can—the most you’re able to eliminate in between the concept and the working system, the better. Everything else is, it’s waste, but it might be necessary waste depending on the situation. So in our case, if [the designer] can go right to creating the user interface in HTML and then I just code stuff to go to it [he means go with it or attach to it], yeah, that would probably be the ultimate. But we can’t. So we communicate through these images, and that’s working for us. (Jack)

Jack describes how meetings with himself, the designer and the president

transpire:

Um, usually the first 5 or 10 minutes [in a working meeting] is, joking around and not being very productive, before one of us says, OK, we gotta get at this. But after that there’s a lot of—I don’t know how you describe it, it’s a very creative process in the sense that, one suggestion will spur an idea, which spurs 5 ideas, which spurs 25. And, that’ll go—if you—if we blocked it as an hour, you know, the first 5 minutes is joking around, then we start—kinda get on some stuff, then we get through this, I guess brain storming is as good as any term for it, ah, and then we kinda have to spend the last 10 minutes kinda reigning everything in and list some core ideas of what we need to do. (Jack)

Jack presents himself as a *pragmatist*, “someone who can set aside one ideal to pursue a lesser, more achievable ideal.” In his interaction with others his aim is to help the team to establish and meet achievable software goals).

Training advocate, 17 years experience. Eric is a learning technology consultant.

Eric is charged with overseeing the development of a Web-based course.¹⁾ His personal goal is to design the best, up-to-date, efficient course possible, and to break a “bad habit” of the design team, which is conducting a learning needs analysis rather than a training needs analysis. He wanted to change the approach the designers take on every project.²⁾

Eric's workplace is unusual in terms of their view of user interface responsibilities.

Designers determine the overall structure of the course and the content, while the developers settle on visual images, interactional elements (buttons, input boxes etc.) and flow of control within a page of the course (e.g., within a page of multiple-choice questions). This was so odd that I began to think that maybe this was a case of developers who were user interface designers and designers who were really content designer, except that it was the designer who determined the overall course direction, structure and type of Web page. Eric's case was also unusual because he was very skilled in the domain of the product he was developing (i.e., learning and training). ³⁾ His view of contradictions is that they are roadblocks to achieving his goal of selling important training concepts and techniques. ⁴⁾ His view of artefacts is that they help to convey concepts and give clients good examples of training techniques so that they can be used in the course the clients needs to design. Eric, more than any other developer was heavily involved and opinionated about the design work for the product and he also was unique because he had built several tools to ease the design and development tasks, such as collections of exemplars for the designers and templates for the developers. Eric had seventeen years of experience in his profession.

⁵⁾ Eric had no issues with the designer he worked with on the project; in fact he got along very well with her and respected her skills. However, in general he finds that designers are not up-to-date, especially with respect to Web technologies, they are not efficient, they conduct a learning needs analysis and not a training needs analysis, and they do not use the concept of "free fail" (i.e., learning by making mistakes), which he believes is an important training concept. The Web designer he worked with was open to learning and changing, and this probably explained his very good relationship with her. ⁶⁾

Artefact-wise, he shares with designers a site of sample Web pages that communicate possible visual layouts and flow but contain no content. The designers use these to explore possibilities and to communicate with the (subject matter experts) SMEs and the developers. He also conducted a meeting to demonstrate to a client and the designer how a training analysis is conducted. He also readily shares the URL to the course Web site while it is under development so that the client and designer can see the status of the build. All of these uses of artefacts promote an approach to Web course design that is training-based rather than learning-based, an indistinguishable distinction to many, but an important distinction to Eric.

Below, Eric describes interactions between himself, the subject matter expert on their project, and the designer. He is opposed to conducting a “learning needs analysis”

Eric: The SMEs, that’s where the expertise is required to be able to answer those questions, those [course] needs. [...] To do a learning needs analysis they’ll often do a meeting, by example, with various users; [the SME], new user, or old—or experienced user—depending on what you need to aim at. And they’re gonna do like a week of ah, user needs analysis.

Interviewer: So you’re not involved at this point?

Eric: We are sometimes because we can help, but usually it’s not our role. Our role will be more, when it comes to the design and [...] solving the [...] need—then we’re a bit, we can be more involved. We’re trying to be more and more involved [in the early stages when needs are determined] because [...] we have a very lot of expertise in our group, so. And that’s where sometimes we can help guiding the ah ... and we’re refocusing the ah—we have a better sense of Web. A Web-sense. That’s where we’re stronger a little bit. So we will help the client understand Web-sense and the designer to sell the idea that the Web is different than a paper base or a classroom, by example. And we have couple of points, like we need to, ah—Web concepts need to be understood by the client. The ah—by example, the difference between engagement and, ah, interaction needs to be understood. And ah, the efficiency of the writing, how it has to be more efficient when it’s on the Web because of the size and the, ah, the self-taught [inaudible word]. So those are little points that, that’s pretty much our role, it’s to sell—that’s one of our role, it’s to sell that—those—concepts to the team.

Interviewer: So then you are not separate from the SMEs? You interact with them as well?

Eric: It would depend sometimes on the designer too. Some designers [feel that interacting with the SME] it's more the role of the designer. So sometime the designer rather [then I] takes control. But if they're more open-minded, I would say, or up to receive other ideas, then they will involve us a bit more. And then it, I think it creates a better—a more complete environment in terms of research for the needs of the course.

Interviewer: So what happened in this case?

Eric: In this case [the designer] involved me a lot. So I did more in that course than usual, I would say, even.

Interviewer: So you did meet with the SMEs?

Eric: Yeah. Yeah, I even sold them a design because they [the SMEs] were not understanding well the training concept of course materials. [...] And that's where I kicked in a bit more because [the designer] is not necessarily experienced—she wasn't experienced enough in the variation between both [learning and training]. Where us here, we've been trainers, then computer-based training, then Web-based training, so we saw them all. We saw the evolution of training.

Eric presents himself as a *training advocate*. As a training advocate he sells training methods and also basic training concepts to his clients and his fellow designers.

Historical anomaly, 15 years experience. Steve was one of three designer/developers in his organization. His background was in public administration, but he was an experienced developer. He was part of a large team that was completely redesigning and redeveloping legacy software that would be used by thousands of people across Canada. The organization had three designers and scores of developers and the three designer/developers who were positioned “between” these two groups; The designer/developer's role was to implement the visual and interactional aspects of the user interface, leaving the design work to the designers and the back end to the “real” developers. No other organization had such a role.¹⁾ His personal goal was to bring the designer's designs to life, to provide some pushback on the designs, and to produce reusable XAML code; reflecting goals that are part-designer and part-developer.²⁾ He believes the designer specifies the UI through interactions with the analysts and the subject matter expert, but that the designer/developers and developers have valid opinions

to offer about the design, thereby negotiating a role for himself in the work of the organization (officially he is just supposed to write user interface code).³⁾ He is very aware of, and accepting of, contradictions or tensions in the workplace—one major one that he experiences is that developers do not accept the designer/developers. He says developers see the designer/developer as “other,” a “visitor,” an “unwelcome intruder,” or as “code owners,” the latter an indication of how the developers are resentful of the control the designer/developers have over certain code modules. Clearly Steve sees the role of the designer/developers is a difficult one on both fronts. He does not have a clear right to design because he provides pushback only, and he does not have a clear right to code because developers resent his presence.⁴⁾ He resolves the conflicts that he can immediately through face-to-face chats, which occupy the better part of his day. He describes artefacts as aids for communicating, reflecting how he uses the bulk of his time, and also his strongest skill which is working with people. He mentions several key artefacts. Status reports go to managers, project plans come down from the managers, sketches support the communication process, as well as express design ideas and answer questions. The most frequently-mentioned artefact was the PowerPoint slides of screen drawings produced by the designers. These he used frequently in conversation with everyone.

⁵⁾ Steve had a very good working relationship with the designer. The only difficulty or tension is that he is reluctant to express an opinion on her designs, fearing he is treading on her territory or the analyst’s territory, or overstepping his role as the user interface developer.⁶⁾ The key artefact the designer shared with the designer/developer is the PowerPoint-generated drawings of the screens for the application. He shared his pixel-perfect reproductions of these drawings with her that he created using a graphical

user interface builder. There was some tension between his artefact and hers because although his looks exactly like hers, it turns out this is not really a requirement, but she has no efficient means of telling him what he may or may not vary when using her drawings to create the user interface.

I asked Steve about his unique role as designer/developer on his project. He explained that his role was transitional and existed only because the designers did not know how to use the tool he uses in his work to “bring the designs to life.”

Interviewer: Yeah, OK. So what role do you and others play in the design activity in this particular project? And, I’m really interested in how you see your role because, to be honest, when I came here I couldn’t decide if you were a designer or a developer, and I don’t know if you have any clear views on that.

Steve: Well [...] when I was hired. And, that’s kind of a good point of view because somebody would be saying: [...] how do we justify paying this money to get this person on board? [...] It was a hybrid kind of description: designer-slash-developer-it was kind of a new role. [He then describes a sequential process where managers, business analysts, designers, and clerks determine what should be built and the designer has expressed this by building a PowerPoint mock-up.] And then we [the designer/developers] come in—and when I say we, my role is then to bring these very specific designs to life. [...] Using this tool [...] it’s a kind of design tool. And I say a kind of design tool because it’s a hybrid product [because it requires both designer and developer skills] [...] called Expression Blend [...] which produces a language known as XAML. It’s Microsoft’s new screen-building tool. And this new tool could be used by a designer. It was intended to replace or compete with products like Adobe, so a designer like [name withheld], in theory, in Microsoft’s opinion, could actually use this tool to do their designs. And it’s possibly because it’s a new product and there hasn’t been enough training on it and not enough people know how to use it. So we have a transitional situation where just during this time, ah, people like [the designer] haven’t been trained on it and perhaps in the future she would go straight from her design straight into screen building. [...] So in a way, ah, my role is to be a bridge between the pure developer (who is taking the screens as we finish them in this XAML state and connecting them with the data so that [...] the application can be useable) and the designers who are in fact creating these designs in the first place. So we have this kind of hybrid role.

Steve presented himself as a *historical anomaly* in the sense that he saw himself as a person who did not occupy a well-recognized role in the project or even the world at large. In his interactions with others he was always aware of his hybrid role.

Practitioner, 10 years experience. James, who is a software developer, worked on a project to redesign a graphical user interface application so that it would have a Web-based, user interface. James is admired and respected by the designers in his organization. He has strong views on how development work should be conducted and emphasizes rigour, methods and accountability.¹⁾ His personal goal was to build the application specified by the designer, to improve processes by building tools (he has made significant project-level innovations in this area), to build maintainable software, to create a successful product, and to mitigate project risks.²⁾ He believes neither the designers nor the developers have the largest impact on the user interface because both must answer to the financial analysts who hold the real power.³⁾ In terms of tensions on a project, James distinguishes between that which he can control and that which he cannot. James's view is that a developer should control the process and be disciplined about the process because it is something he can control. Other issues that are out of the developer's control are to be accepted such as difficult problems or unexpected implementation issues. Another larger issue out of his control and which James accepts is the "culture" of the company, which favours sales over development, and development over design. This he believes is unlikely to change in the near future. He sympathizes with the designers in his organization who he feels are obviously disregarded.⁴⁾ With respect to artefacts, his view is that to use artefacts well is a discipline that he enforces in his projects to combat "folklore" (meaning developers making false assumptions), chaos (developers making conflicting assumptions) and to get better project outcomes.

⁵⁾ For James the key contradiction between the designer and developer centers around the creation of the project specification, which is a text document. ⁶⁾ James feels it is very important to be extremely clear when specifying the user interface because as a practitioner he needs clear directions to perform well and because he does not like sloppy work. He does not believe that screen images describe an application. He believes that although an image may convey 1,000 words it is not clear “which 1000 words the image conveys.” In fact, he believes in fully specifying *everything* to the extent that he thinks that two developers should never talk to each other without recording important decisions or agreements in textual form.

James describes his relationship with the designers indirectly by describing how he manages interactions between his developers and the designers in his organization. His stance is that you respect the professional skills of others.

Like when the GUI team gets involved with a development project that says, in so many words: “This is how it should be done, this is how the work flow should be, this is what the GUI should look like.” There’s often a strong push back from the development team that doesn’t wanna do the extra work. Or that believes that it’s just unnecessary stuff for them to worry about. Or that maybe honestly believes that, you know, they’re just as good at deciding what a GUI should look like and feel like as the people who have training [the designers]. [laughs] Like I mean, on our team, I’ve—I know I’ve told people, I’ve told developers on our team, like: “You can’t put a text label into this application. You can’t put a dialogue into this application. You don’t know what we should be saying to the user.” Right? Like, if you’re doing that, you’ve exceeded your authority, right? And it may seem very constrictive, right? Because you’re saying like, “You can’t decide what a colour is, you can’t decide where things go, you can’t decide what the text is—none of those things are your decision,” right? And I think that for developers, they get a little bit—sometimes—they get a little bit, on an ego level, feel like they’re being constrained to doing very cut and dried tasks because they have no say or control over the way the application looks and feels. But, I’m not sympathetic. (James)

James presented himself as a *practitioner*, or a person whose practices are professional and who expects professional practice from others. When James interacts he

conducts himself in a professional manner and he expects a high standard of work in return.

Problem solver, 10 years experience. Damian is a software designer (designer is being used in the software developer's sense meaning that Damian designs software programming objects to implement the user interface). In his organization he is charged with designing a software framework for future features and also implementing the Web interface for a very complex application that will require designing a windowing system, something that is usually a fixed and predetermined aspect on most projects.¹⁾ His personal goal is to create a framework for GUI development, and to reproduce previous functionality in a Web environment (i.e., the big problem he must solve is how to develop a general solution for future GUI development while showing its viability by producing the first release of the product).²⁾ He says the designer is responsible for designing the user interface and as such the designer is a part of the team. However, usually when Damian says "the team" he means himself and the other developers assigned to the project and the occasional short-term team members such as the designer. He resists calling the Scrum team (to which the designer belongs) "the team" because he resists being a part of that team because he is not yet convinced of its benefits. Damian says the user interface design is "not interdependent" with the work of the developers, meaning his work is dependent on the designer's work and that he cannot do his problem-solving in isolation from the designer.³⁾ He views contradictions or tensions as problems to be solved and he says⁴⁾ artefacts help with the task of problem-solving and building.

⁵⁾ For Damian, creating the right platform for user interface development work is of primary importance.⁶⁾ Before any design work had begun Damian presented demos of other applications to the designer to show him what is feasible in the development

environment the developers were planning on using (there was a lengthy selection process) and in this way he hoped to influence the designer's thinking so that the concepts (or design) he comes up with will be ones that make it easy for Damian to design (in the software developer's sense) from. Essentially, he wants the designer to know what is possible and to design within those constraints so that his problem is easier to solve. He also wants to show examples of modern designs so that the design will be modern. In his interactions with the designer Damian worries the designer's ideas may not be the latest, they might not be feasible, or they might break a "philosophy" of the software system, all of which would make his work harder. Damian sees himself as a person with difficult problems to solve, one of them being how to do what he has to do in the context of Scrum, the development process being tried on his project. He has two issues with this process. One is that it requires him to estimate time on task accurately and he finds it very difficult to do this because of the novelty of his work, and also the invisibility of his work to everyone on the Scrum team except developers. Second, because of the pressure on him he is not interested in knowing about the problems the designer (or any nondeveloper) faces on a day-to-day basis because he feels he cannot really help and it takes away from his own productivity. Daily meetings where all types of problems are aired are a requirement for Scrum and these take up about 45–60 minutes of his time every day including preparation and follow on tasks.

Damian describes his role relative to the role of the designer. Damian uses the word design mostly in the sense of software design and here he explains how the designer's work is simple design and his software design work is problem-solving.

Damian: It is useful, yeah I mean, the [software] design artefacts [he means the software specifications the developers write], yes it does...because it'll give us an idea as to how complex things [the user interface windows] are.

[...] But that's for [software] design. Like, but [name withheld] is not involved in that kind of a design, right?—[...] he's involved in a very simple design. Like, you know: forms, flow of this. Whereas we are involved in framework [design], which is problem solving.

Here Damian describes his implementation task, also in terms of problem solving:

Damian: when we have a feature ... Right now the way it's working is that he [the designer] attends a meeting to understand what we're trying to implement. And then he comes up with concepts and workflow, ah, of that feature, a sort of user interaction. Um, and those are—like, he just comes up with basically concepts and that are presented in a meeting. And then we give our opinions and then it gets it finalized and then we build the—implement the form or this [whatever] or anything. [...] So it kind of relieves us from the whole user interaction, right?

Interviewer: Yeah, 'cause—

Damian: Right? We are mainly focused on solving problems and implementing features.

Damian presented himself as a *problem solver* in the sense that he is someone who considers a situation, matter or person that presents perplexity or difficulty, especially dilemmas that defy satisfactory solutions. In his interaction with others he sees his role as finding satisfactory solutions to perplexing and difficult software problems.

Catalyst, 7 years experience. Kylie was the only female software developer in my study. She was the lead designer on a project that involved developing a feature for an established graphical user interface product. ¹⁾ Her personal goal was to produce the feature in a timely manner and to produce something that's usable and consistent with the feel of the product; in adopting usability and consistency goals as her own, she has overlapping concerns with the designer. ²⁾ She believes the designer produces the initial proposal and the developer and other stakeholders shape and alter it (sometimes significantly) with their responses. ³⁾ Her view of contradictions or tensions is that they are issues to be addressed. She told me a number of stories about problems that occurred during the implementation of the feature including that the design as specified could not

be built due to the choice of terminology, the choice of controls, multilingualism problems, consistency issues with the rest of the product, and unspecified details in the specification including a lack of rationale or explanation for the decision choices. Kylie is a developer who provides strong critical feedback to the designer. The designer she worked with was grateful for the feedback and responded rapidly by altering the design.

⁴⁾ She describes artefacts as catalysts for change. For instance, she says that emails are useful for distributing designs widely to raise issues and elicit feedback, and she told me a story about using email to initiate a significant change to the design when she was working from home one day. Documents (sometimes as email attachments), she says, are for addressing detailed issues.

In terms of her relationship with designers, the distinction between the various types of designers is not important to Kylie (there are at least four different types of designers in her organization). She sees their concerns as being the usefulness, usability, and “look and feel” of the product. She is unclear on designer processes (like what their design review process is), and she finds them to be document-centric. ⁵⁾ She is very happy working with the particular designer on the feature and her work is satisfying in this regard. Not only is he a good designer in her opinion, but he is also open to hearing her suggestions and acting on them. When implementing the multilingual aspects of the feature she required numerous changes to the design to accommodate phrases of different lengths in different languages. Kylie also provided others opportunities to be a catalyst by making her work easily available. She worked very hard to build an environment so that the designer could see her work on a daily basis as she implemented the feature, perhaps inventing a novel approach along the way that other projects could copy. ⁶⁾ The product itself was a key artefact she shared. She appreciated the many artefacts that the developer

produced, especially his PowerPoint presentation and the design specification, both of which provided Kylie with opportunities to respond to his designs.

Here Kylie describes how she responds to the designer's drawings.

Um, well, you know, it's weird. ... When you ... you know, you can just draw something up and you think, ah that'll work. But then once you see it, you know, you can, you know, pick other issues with it, right? So yeah, it generated discussions and it—And, because he was able to send that around to a wider group of people than, say, you know, a piece of paper, it's more clear of what you're going to end up with and we can get more feedback (Kylie).

And here when she tells a story of how a design was usable, but not consistent with other aspects of the software, which she raised as a problem.

He [the designer] had a bit of a lesser focus on consistency, I would say, and more on—more on usability, and, I mean, I'm sure he was sensitive to time as well. But I think he kinda had the idea that, um, you know: Let's just work ()—let's just worry about making the one dialogue useable. And if it's not consistent, well, hopefully we'll eventually have time to make, you know, the rest of the product more useable and therefore consistent. So, you know, it's a bit of a different way of looking at it. Do you know what I'm saying? (Kylie)

Kylie presented herself as a *catalyst* in the sense that she saw herself as a person who caused important changes to take place. She acted as a catalyst within the team to ensure changes in the product were made so that it was usable.

Assembler/builder, 3 years experience. Jason is a game developer. He was one of two game developers in his organization and worked with several designers and for this reason this developer was unique. ¹⁾ His personal goal for the project is to contribute to the design and to produce or build the game. ²⁾ He believes there is a shared responsibility for the design of the game, particularly in the beginning stage where everyone is a designer, including himself. ³⁾ With regard to tensions on the project he sees these as natural, and a consequence of human nature. ⁴⁾ Jason is results-focused and artefacts such as gaming concepts, detailed arrangements of screen elements, a sense of the game feel,

and a description of the interaction are elements that he needs to bring together or assemble to build the game. For Jason, artefacts play a significant role in fully specifying the game that he must build. He says that “every level of a design has a different demand on what artefact can be used” and explains that interaction stories are a good starting point for introducing a gaming concept, but addressing arrangement issues of the “game feel” requires a sketch, and a prototype is required to get at the interaction between the gamers and the game. Jason also receives the individual work of all the designers via artefacts that are emailed to him and he assembles and builds these into the game, which he continually reproduces on a daily basis. Jason always prefers to talk to communicate with others on his team and says that drawing is a last resort for him when his talking will not work. Key artefacts for Jason are the game that he produces, working meetings and chats (meetings and chats are experiential, structured ways of accomplishing interactions).

Jason was working with four designers simultaneously and sees each designer as unique, but he says they all communicate with him through artefacts which they submit to him for integration into the product. He says they are each concerned about a different aspect of the product.⁵⁾ He discussed many tensions with designers including nongamey feature requests which he intuitively felt would be design mistakes; feature requests that would take too long to build; contradictory assumptions between artists and content designers which are preventable and time consuming to resolve; and contradictory assumptions about features between the designers or between a designer and the future gamer who is not going to understand the game. All of these tensions impact his assembling and building work because it is largely the designers that determine what he assembles and what is built.⁶⁾ In his workplace, the game itself, which this developer

creates, is at the center of the work because it makes visible the false assumptions, missing elements, poorly designed elements, or a lack of fit between the work of two designers. Issues about what features to design and the effort required are mostly resolved through talk.

Below Jason describes how he continually rebuilds the game for the design team from a sequence of artefacts that provides increasingly more detail about the game the team is creating.

We want a menu with 5 buttons and, you know, the user clicks on each button and the text changes or whatever. So that verbal description is good for a certain level of that problem, right? And then when you put it in the game, it's no longer, We want 5 buttons and the user clicks it and the text changes, it's how (are) those buttons (arranged). So that could be handled by a whiteboard, right? You sketch out a user interface. And that's that scale of the problem. So now you have, the graphic artist goes and makes these buttons and arranges them and I code them, right? So now there's the question of how does it feel when a user's using that menu? The whiteboard can't tell you, you know, what the button does when you roll over it. Or what the button does when you click it. Or how the text comes in. So that's something that you need to use the computer for. And you use Flash to code the game, so that you can start getting that interaction. [Here Jason tells the story of himself as the developer who uses Flash to code the game.] So you can—every level of a design has a different demand on what artefact can be used. [Jason associates artefact types with problem types here.] Does that make sense? (Jason)

Jason presented himself as an *assembler/builder*. As an assembler, he 'brings [things] together into a whole'; as a builder he is 'someone who develops or gives form to something according to a plan or process'. In his interaction with others he assembles parts and then produces a new whole, which he then shares with the team.

Appendix G: Tables of Meetings at Talia, Telespace, and Quadrow

Meetings at Talia

Mtg #	People	Key Artefacts used in meetings	Summary of the meeting
1	The Config-it team		The developers present applications developed in different development environments, pointing out features of the environments.
2	The Config-it team	A usability evaluation of the current software	The designer presents a usability evaluation of the current version of product. The product manager wants something like product XYZ, does not need it to be backwards compatible with current Config-it version, 12 or more clients are anticipated, there are no guidelines for object attributes, discussion of possible UI styles occurs (menu, tabs, text,...), designer asks is the product about states, monitoring or alarms? Talk of structure (meaning what windows would be required), talk of look and feel and what is and is not possible with different development environments.
3	Product owner, designer, developers' manager, current product expert (4)		The attendees "go through" all of the user interface of the previous version of the product, and discuss what to drop. The main goal is to "remove elements that come from the original version of the product," which has now been radically transformed.
4	Developers' manager, designer, designer's manager (3)		Designer presents first mockup. They discuss: initial view for a certain role, layout of windows, filters, information windows. There's a discussion about views for different viewers. They talk about what is and is not hard to do in the development environment. They "dis" wizards, floating windows, stacks of windows. Object wells and filter shortcuts are put forward as an idea. The designer says he wants developers to get on board with these ideas rather than "workflow" stuff
5	The Config-it team		The second and third mockups are presented by the designer to explore which windowing mechanism to use. The choice seems to be between using object wells and shortcut filters versus having a main window background that is tuned to the role of the user, filters defined by administrators and user-defined filters, wizard entry point and summary information. The designer's manager also presents an image of Talia's new look and feel.

6	Product Team	The designer presents a fourth prototype that uses ideas from another product.
7	The Config-it team	Designer presents the 5th, 6th mockups. These are 10high fidelity renderings of windows and one drawing of 11a drag and drop operation. Main difference between 5th and 6th prototype is tht there are actions on items in listed objects in the manager window. The 7th prototype was built with Flexbuilder and is of the application embedded in the browser
8	The Config-it team	The designer reports he demonstrated the 7th prototype, an interactive one, at a spring meeting 'unchanged'.
9	The Config-it team	A meeting with this flow diagram about a particular form was held.
10	The Config-it team	A colleague of the designer presents a report on options for designing filters (complex and simple) being considered for another product. Implementation concerns are aired. Stories told.
11	Lead designer, 4 developers, tester, developers' manager, project owner (8)	A review meeting of the requirements specification for the user interface. Interaction style will be AJAX-like. Decide type of application is a content management system (CMS) with various navigation systems (portlets, maps,trees,). Product owner wants a CMS GUI. Requirements. Product owner clarifies there are three interfaces and three groups of users. Hence, need scope of command feature, and have security issues. Developer sketches arrangement of four windows to illustrate perspectives idea, a way of grouping windows for a user group. Suggestion of portlets is made to address security. Interface style to be tabbed with no minimizing, but detaching and reattaching. XY will be used as a model.

Meetings at Telespace

Mtg #	People	Key Artefacts used in meetings	Summary of the Telespace Meeting
3	<p>Designer, designer-developer, developer (3)</p> <p>Note. The designer-developer implements the user interface and was referred to as the designoper in the text.</p>	<p>Sketch of a popup for the feature. sketch of what can be done in XAML. A sketch of the old interface.</p>	<p>The designer says she wants many pop-ups for a reporting feature built. Three alternative designs are discussed. 1) Using a database that has a set of rules about what options are necessary for each report. The designer says there might be 25 different pop-ups. They discuss whether XAML code could solve the problem. They discuss: speed issues (one popup would be faster), They discuss variations to do with the fields. 2) Using a master screen with layers in XAML. The designer says her rules are "no grayed out fields or gaps." 3) XAML stack panels and a binding mechanism. The developer checks this could be done at run-time. The designer-developer says he needs a worst-case scenario popup. The developer is interested in the workload for the designer-developer. They go to the lab to select a report and a misunderstanding in terminology is cleared up. The designer-developer sketches and explains a solution. The developer reveals reports have types and they explore how this might be significant. They discuss the field labels and how these will appear; currently these labels are strings in a database.</p>
4	<p>Designer's manager, two designers, three design-developers (6)</p>	<p>The developers' managers' email about important issues.</p>	<p>The designers step through the developers' manager's priorities and identify roadblocks. They guesstimate time to completion. A misunderstanding is revealed about how "not-quite-so-generalizable" the designer-developer's solution is. One issue is that the designers are always struggling to understand the old system because the analysts are describing only additions and not deletions. The developers are pressuring the designers for screens. They say they do not want instructions on screen behaviors. About four features are discussed including the reporting feature.</p>
	<p>Designer, two Design-Developers (3)</p>	<p>The designer's wireframe for the worst case options scenario.</p>	<p>The designer and designer/developer agree that there are 21 potential options and of these 9 will appear in the worst-case scenario.</p>

16	Designer, two developers, three designer-developers (6)		The developer is concerned about the help feature for reports that was available in the previous version of the product. He brings a printout of help messages in a database table to the designer/developer's desk. The three discuss 3 different options for incorporating the tips. 1) Do not do it at all. 2) Do it beside each option. 3) Provide help in one location at the bottom of the window. Along the way, they discuss what is static and what is dynamic and how help is handled generally in the system.
19	Designer, two designer-developers (3)		The designer/developer demonstrates his work so far. The developer says she's going to have to change the wireframe, but after a discussion, the designer/developer decides to work from her spreadsheet.
25	Designer, developer, design-developer (3)		The designer wants to know how exactly the reporting feature is going to work. The developer reveals that there are templates that the analysts produce that are essential to making the report and options appear. The designer reveals restrictions on total number of options. The developer makes sure he will test the most extreme case. Stepping through some screen dumps and simulating user actions answers a question on navigation.

Wireframe printout, printout of option tips stored in a database.

The designer's spreadsheet is now the definitive artefact on reports and options in the new system.

A list of reports and options pulled from a database, the screen dump of the previous system, analyst-designed templates.

26

Developer,
designer-
developer

The interactive prototype.

The designer/developer demonstrates the collapsible popup screen. The demo fails, but pair discuss whether implementing help for options will be hard or not and the implications for the current design.

Meetings at Quadrow

Mtg #	People	Key Artefact	Summary of the Quadrow meeting
1	New developer, database expert, Calcul-it developer, project manager, new developer, developer on another project, interaction designer (7)	<p>Tickets</p>	At the daily standup meeting the Calcul-it project and 3 others are discussed. The issue is whether or not projects are on track.
2	Project manager, 5 developers, database expert, designer (7)	Tickets (see meeting 1)	The Calcul-it project are reviewed ticket by ticket. The Calcul-it team prepares for their meeting with their client.
3	Client, 2 assistants to client, project manager, designer, 2 developers, database expert (8)	<p>Client's paper mockups and tables of scenarios.</p>	The client wants some changes to the Calcul-it prototype. She has drawings with new screens and new buttons and a dozen scenarios.
4	Project manager, designer, developers, database expert (5)	Client's paper mockup, database schema	Discussion of implications of the client suggestions. The project manager decides they need to plan a meeting to discuss technical implications. How much more complex is the new design? Should certain conditions be considered within scope or not?
7	Designer, project manager, database expert (3)	Electronic mockup on plasma screen, client's paper mockups, excel calculator engine simulator, high fidelity mockup, whiteboard sketches, tables drawn on whiteboard to facilitate	The group reviews changes and implications, particularly for feasibility.

calculations			
8	Project manager, designer, developer, database expert (4)	Sketches of a new web page for the application. Some scenarios to test the design.	Calcul-it changes are reviewed, especially with an eye to implementability. The client has made the work harder by requesting one very large page, instead of a set of pages. The project manager says they need more concrete scenarios and not just toy data.
9	Project manager, designer, database expert (3)	<p>Client paper mockups, concrete scenarios, rules for the calculating engine, examples of how data would be entered in the Calcul-it interface, a simulation of the calculation engine.</p>	<p>The team is deciding if the client's proposed design is workable.</p> <p>Artefacts: Client paper mockups, concrete scenarios, rules for the calculating engine, examples of how data would be entered in the Calcul-it interface, a simulation of the calculation engine.</p>
11	Developer, designer (2)	Electronic mockup	The designer and developer talk about one of the fields on one of Web pages and whether or not the page could be broken up to make the developer's work easier. There's other discussions about a client change that has added complexity.

Appendix H: Details of meetings that were video recorded

Note. Underlined phases below were unusual and not included in the model.

Jaba meetings video recorded and analyzed:

- A: (2 hours) In a scheduled meeting, two designers and a developer evolved the gaming prototype.
 - 1. Presenting, 2. Simulating, 3. Assessing, 4. Simulating, 5. Assessing, 6. Simulating, 7. Assessing, 8. Simulating, 9. Considering implementability (formerly creating) 10. Assessing, 11. Simulating, 12. Closing (formerly next steps)
- B: (1 hour) In a scheduled redesign meeting with the developer, the graphic designer, the usability specialist, the game designer and the learning/content designer.
 - 1. Opening, 2. Simulating, 3. Assessing, 4. Enacting, 5. Simulating, 6. Assessing, 7. Simulating, 8. Assessing, 9. Simulating, 10. Modeling, 11. Simulating, 12. Exploring, 13. Simulating, 14. Reviewing, 15. Simulating, 16. Closing.
- C: (roughly 30 minutes) In a scheduled meeting, the developer presented a prototype of a gaming element to the game designer. He had both designed and implemented this element that he was proposing for integration into the game.
 - 1. Opening, 2. Presenting, 3. Exploring, 4. Simulating, 5. Considering 'implementability', 6. Exploring, 7. Considering 'implementability', 8. Closing.
- D: (5 mins) In an impromptu meeting with the game designer, developer, content designer and graphic artist consider their future work
 - 1. Opening, 2. Considering 'implementability', 3. Closing.

Quadrow meetings video recorded and analyzed:

- A: (1 hour 30 minutes) In an impromptu meeting, the product owner, designer, and database expert tested a mock-up of the interface provided by their client.
 - 1. Opening, 2. Simulating, 3. Simulating, 4. Assessing, 5. Simulating, 6. Simulating, 7. Simulating, 8. Simulating, 9. Simulating, 10. Assessing, 11. Simulating, 12. Enacting, 13. Closing, 14. Assessing, 15. Enacting, 16. Closing.
- B: (15 minutes) In a chat at the designer's desk, the developer and designer discussed an issue to do with the model for the database that affects them both.
 - 1. Opening, 2. Simulating, 3. Assessing, 4. Exploring, 5. Closing.

Telespace meetings video recorded and analyzed:

- A: (20 minutes) In a work-related chat at the designer's desk that also involved the designoper, the developer asked the designer how help messages he's found in a database would be displayed in the user interface.

- 1. Opening, 2. Simulating, 3. Exploring, 4. Closing
- B: (5 minutes) In a work-related chat, the designer asked how the developer knows exactly what the user's prompts are when the user is tailoring reports. Examples of prompts are 'which month?' and 'number of copies'. There were scores of prompts.
 - 1. Opening, 2. Enacting, 3. Simulating, 1 closing.
- C: (10 minutes) In a work-related chat, the developer had a question for the designer about restrictions regarding the arrangement of the values associated with report prompts.
 - 1. Opening, 2. Simulating, 3. Enacting, 4. closing
- D: (5 minutes) In a work-related chat, the designer demonstrated his popup window for report attributes to the developer.
 - 1. Opening, 2. Simulating, 3. closing

Talia meetings video recorded and analyzed:

- A: (45 minutes) In a scheduled meeting, the Talia team discussed what not to port to the new user interface in a walk-through of the current product.
 - 1. Opening, 2. Enacting, 3. Enacting, 4. Enacting, 5 closing.
- B: (1 hour) In a scheduled meeting, the designer presented several designs for windows he is designing. The developers and others on the team responded to these designs.
 - 1. Opening, 2. Presenting, 3. Assessing, 4. Presenting, 5. Assessing, 6. Presenting, 7. Assessing, 8. Simulating, 9. Presenting, 10. Assessing, 11. Presenting, 12. Assessing, 13. Presenting, 14. Presenting, 15. Assessing, 16. Presenting, 17. Presenting, 18. Presenting, 19. Assessing, 20. Closing.
- C: (1 hour) In a scheduled meeting the designer presented another design. The developers and the others on the team responded to these designs.
 - 1. Opening, 2. Presenting, 3. Presenting, 4. Assessing, 5. Presenting, 6. Assessing, 7. Simulating, 8. Presenting, 9. Assessing, 10. Presenting, 11. Project planning, 12. Simulating, 13. Presenting, 14. Assessing, 15. Presenting, 16. Presenting, 17. Presenting, 18. Assessing, 19. Closing.

Appendix I: Evidence from this research supporting the existence of contradictions in the developers' roles

Contradiction reflected in the developer's role			Evidence for contradiction in the developer's role
A developer may consider him or herself to be like a worker in another discipline,	◀▶	but this analogy will necessarily have limitations.	Developers did not identify as engineers, they constructed unique identifies (see Perspective 3).
A developer needs visualizations to carry out his role,	◀▶	but none are fully adequate.	Developers could not work without a visualization of the software, so many switched rapidly between visualizations when working to get a more complete picture of the software's state.
A developer's methods for solving software problems are not without value,	◀▶	but they do not solve the problem of creating the software.	Methods were valuable general-purpose frameworks for organizing development work, but no software problem was ever solved by asking, What does our methodology say to do here?
A developer does not have a science to help him/her solve problems,	◀▶	but rather draws from multiple sciences or no science.	Not one developer was observed directly using mathematics to help them program, but I did see them using the best practices of their profession.
A developer's role may be to build software to specification,	◀▶	but their ability to do this in all matters is limited.	Only one developer I observed worked from a specification and he frequently spoke with the designer and analyst for clarifications. Another developer was a co-participant in the construction of the requirements. Another had to deal with rapidly churning requirements. Yet another developer was helping to determine the requirements by building the product. The relationship between the developer and the requirements was unique at each site. Requirements were always an issue.
A developer's primary task (coding) is highly rational,	◀▶	but other aspects of his or her task cannot be approached using reason alone.	There was much evidence to suggest that coding is a rational task. For instance, one developer conducted experiments to select a tool set. Another protected himself from churning requirements by coding stable parts of the software first, and relied heavily on testing to ensure program correctness. But two of the developers were more reliant on people skills. One was given the task of improving communication between designers and developers. The other made his workplace environment productive by integrating the work of others in a particularly helpful and patient way. Almost every work product he was given by a designer was faulty in some way, but he quietly went about fixing it with the designer on the side, and just made everything work seamlessly.

Appendix J: Evidence from this research supporting the existence of contradictions in the designers' roles

Contradiction reflected in the designer's role			Evidence for contradiction in the designer's role
A designer expects to design novel interfaces,	◀▶	but much of their career is spent redesigning and adapting interfaces to new devices, platforms, user groups or environments.	<ul style="list-style-type: none"> ▪ Two of four sites were working on projects triggered by a change in technology. One went from a GUI interface to a Web interface. Another went from a GUI interface to a touch screen interface. ▪ One project completed, but never used because it was an aid to learning about a technology that changed during the course of the project, making the software dated before it was released.
A designer will take a disciplinary-based perspective to their work,	◀▶	but they may not feel a connection to the wider field	<ul style="list-style-type: none"> ▪ One designer (whose background was in industrial design) identified with the interaction design community and began a Google group for interaction designers. ▪ One designer (whose background was in psychology) posted a sign to let everyone know she was a user needs analyst, usability specialist and interaction designer; she was an executive for a usability professional's organization. ▪ One designer (whose background was in learning and psychology) described herself as a learning and content designer and was active in a local chapter of CHI. ▪ One designer (whose background was in industrial design) described himself as an industrial designer with training in user needs analysis and usability testing.
A designer may use theories explicitly or implicitly to guide their work,	◀▶	but disparate theories may not lead to a unified design and nothing will ever replace trying and seeing what happens.	<ul style="list-style-type: none"> ▪ One designer used Bloom's taxonomy and a theory of flow. ▪ One used game theory and mathematics to compute complexity of content design problem. ▪ One used card-sorting techniques to simplify a problem ▪ One relied heavily on international standards on accessibility ▪ One incorporated organizational standards for interface elements that would have adhered to many design principles.
A designer may have an idea about how they should design,	◀▶	but may find it challenging to apply or adapt these approaches within multidisciplinary teams.	<ul style="list-style-type: none"> ▪ Mature designers adapt their approach over time (see results in Perspective 3). ▪ All designers were frustrated by lack of access to end users.
A designer expects to design novel interfaces,	◀▶	but much of their career is spent redesigning and adapting interfaces to new devices, platforms, user groups or	<ul style="list-style-type: none"> ▪ Two of four sites were working on projects triggered by a change in technology. One went from a GUI interface to a Web interface. Another went from a GUI interface to a touch screen interface. ▪ One project completed, but never used because it was an aid to learning about a technology that changed during the course of the project, making the software dated before release.

environments.			
Contradiction reflected in the designer's role, continued		Evidence for contradiction in the designer's role, continued	
A designer will take a disciplinary-based perspective to their work,	◀▶	but they may not feel a connection to the wider field.	<ul style="list-style-type: none"> ▪ One designer (background in industrial design) identified with the interaction design community and began a Google group for interaction designers. ▪ One designer (background in psychology) posted a sign to let everyone know she was a user needs analyst, usability specialist and interaction designer; she was an executive for a usability professional's organization. ▪ One designer (background in learning and psychology) described herself as a learning and content designer and was active in a local chapter of CHI. ▪ One designer (background in industrial design) described himself as an industrial designer with training in user needs analysis and usability testing.
A designer may use theories explicitly or implicitly to guide their work,	◀▶	but disparate theories may not lead to a unified design and nothing will ever replace trying and seeing what happens.	<ul style="list-style-type: none"> ▪ Mature designers adapt their approach over time (see results in Perspective 3). ▪ All designers were frustrated by lack of access to end users