Agile Experience:

Communication and Collaboration in Agile Software Development Teams

Elizabeth Whitworth, BS Human-computer Interaction

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Arts

Department of Psychology

Carleton University

Ottawa, Ontario; Canada

September, 2006

© Elizabeth Whitworth, 2006

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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: 978-0-494-18306-9
Our file Notre référence
ISBN: 978-0-494-18306-9

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 protège 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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

This was a qualitative grounded theory study of subjective experience in 'agile' software development teams, which were conceptualized as complex adaptive socio-technical systems. The basic research question was: *What is the experience of being in an agile software development team?* This was further structured to ask: How do agile practices structure and mediate the experience of individuals developing software? In particular, how do agile practices mediate the interaction between individuals and the team as a whole? And what are the socio-psychological characteristics surrounding agile practices and individuals within a software development team? In particular, what agile practices contribute to positive feelings such as motivation and excitement? Twenty-two interviews with agile practitioners were conducted, and analyzed in light of relevant literature using grounded theory. Results include a deeper understanding of socio-psychological phenomena occurring in teams. Social environments supporting commitment to a collective endeavor were seen to be particularly important.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

Deepest thanks to Robert Biddle for his unfailing support and for being so wonderful and smart. Thanks to Angela Martin and Lynn Miller especially for their help in conducting my research. Thanks to the Agile Alliance for providing funding for this study. Thanks to my committee members Janet Mantler and Tzvetanka Dobreva-Martinova, my external advisor Alex Ramirez, and to Judy Brown, Michael Feathers, and Axel Busch for their input and encouragement.

Thanks to my father for making me think, and my mother for her enduring support. Thanks to Sarah and Marta for turning up the good, and for putting up with me working (and sleeping) in the living room. Thanks to Rippin, Pilla, and Matt – because it's all about the pants. Tokyopants. Thanks to the Ottawa breaking community for being a welcome distraction. Thanks to members of the HOT Lab for your support and shared experiences. Thanks to Marilyn (and Scott) Tremaine for being a true inspiration.

Thanks finally to my participants and to the agile community for providing data for this study; particularly to those of you who illustrated the depth of thought, care, and passion that was the initial inspiration for this research.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Appendices

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

AGILE EXPERIENCE: COMMUNICATION AND COLLABORATION IN AGILE SOFTWARE DEVELOPMENT TEAMS

Introduction

'Agile' is the label for a new variety of adaptive software development methodologies that have been gaining momentum in the software engineering discipline. Agile processes are most often offered as a solution to the problem of vague and changing product design requirements (e.g. Augustine, Payne, Sencindiver, & Woodcock, 2005), and are held in contrast to more prescriptive processes for software development. Agile methodologies can also be seen to address an additional problem space; namely, the fact that traditional software development processes do not fully account for the presence of humanity and human interaction (Cockburn, 2000). This thesis defines agile software development teams as complex adaptive socio-technical systems. It involves a grounded theory study of subjective experience in agile teams, as a step towards a deeper understanding of human action and experience during software development.

What is Agile Software Development?

Agile methodologies provide a structure for highly collaborative software development. Developed in the 1990's, the adaptive methodologies were formulated by and for developers in reaction to perceived deficiencies in conventional 'top down' or 'plan driven' methods. Commonly associated with 'lean' engineering (e.g. Poppendieck & Poppendieck, 2003), agile software development closely follows the flow of business value, with a focus on activities that directly contribute to the project end goal of quality software. The agile manifesto (Beck et al., 2001), a guiding force for agile practitioners,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

was created by 17 influential figures in the nascent software development movement, and is outlined as follows:

> Manifesto for Agile Software Development
>
> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> *Individuals and interactions* over processes and tools
>
> *Working software* over comprehensive documentation
>
> *Customer collaboration* over contract negotiation
>
> *Responding to change* over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

While the originality and general applicability of the technical and engineering aspects of agile are up for debate (e.g. Boehm, 2002; Larman & Basili, 2003), agile methods are distinctive in that they explicitly integrate behavioral and social concerns into engineering practices. Indeed, the 'people' focus of agile methodologies is singled out as an essential factor in their success and growing popularity (Boehm & Turner, 2004; Cockburn, 2002).

Agile methodologies are specifically designed to facilitate communication, collaboration, and coordination within a dynamic team environment. This includes practices such as shared team rooms to encourage frequent informal communication, informative workplace environments to allow ubiquitous information dissemination, and rapid feedback cycles through which a software product is evolved incrementally in close partnership with a customer representative. Rather than adhering to traditionally long

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

periods of upfront requirements gathering and design before software production, agile teams elicit feedback early on in the process, and deal with the complexities of software development by practicing rapid iterative development from project inception.

Agile practitioners 'fit the process to the project,' restructuring work in reaction to the needs of the current project, and take pride in rapid flexibility (agility) when faced with the vague and changing needs of today's business context. Development generally occurs in small teams of 10-15 people, who work together as a cohesive whole.

Motivation for Research

In the 2nd edition of *Extreme Programming: Embrace Change* (Beck, 2004), a seminal book in the agile movement, Kent Beck adds to his original work a new first sentence – "Agile is about social change." Agile or 'lean' methodologies can be seen to be offered as an alternative to traditional software engineering processes, much in the way that lean manufacturing (Womack, Jones, & Roos, 1990) was offered in place of the assembly line – with significant human and social gains in addition to business value (Highsmith, 2002; Poppendieck, 2002).

There is a growing body of practitioner evidence to suggest that participants in agile team environments find the experience particularly rewarding; more so than most other software development environments (e.g. Beck, 2004; Jeffries, 2004). A survey by Cockburn and Highsmith (2001, p. 3), for example, found that agile methodologies were rated higher than other methodologies in terms of morale, while Goebel (2002, p.12) found in an informal poll that people who worked in agile teams were reluctant to go back to old methods of development. Although the 'hype' surrounding agile methodologies (Stephens & Rosenberg, 2003) is likely a strong contributor to the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

enthusiasm observed, there seems to be some basis for the common association of agile practices and teams with the idea of 'project chemistry' (Nicolini, 2001) or positive 'team climate' (Anderson & West, 1994) that can contribute to a high performance (Goebel, 2002).

Attempts to explore socio-psychological phenomena related to agile methodologies in more detail, however, were stymied by the lack of basic research into the lived experience of individuals in agile software development teams — or any other kind of software development team, for that matter. While pockets of literature surrounding software development methodologies contain strong references to socio-psychological issues, such as ego, well-being, control, and team conflict (e.g. Bullock, Weinberg & Benesh, 2001; DeMarco and Lister, 1999; Weinberg, 1971), software development literature was seen to be primarily practitioner based, concerning the practicalities of software construction, software development processes management, and the hurdles of making it all work within a business context. Academic studies were further seen to maintain a focus on specific behavioral or cognitive effects related to technology or process.

Studies into 'the psychology of programming,' for example, were seen to focus on cognitive processes and mental representations involved in detailed programming and software design (e.g. Bednarik, Myller, Sutinen, & Tukiainen, 2005), or on particular agile practices, such as pair programming or shared team rooms (e.g. Chong, Plummer, Leifer, Klemmer, Eris, & Toye, 2005). Project management level research into individual psychology during software development was also seen to focus primarily on the effects of tasks, roles, processes, and tools on measurable behavioral outcomes such as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

productivity and attitude (e.g. Rubin, 1993). Such research seems to exclude a deeper understanding of socio-psychological phenomena, such as motivation, that are commonly associated with the agile team environment.

While the 'software psychology' movement in the 1970's (e.g., Miller, 1974; Shneiderman, 1980; Shneiderman, Mayer, McKay, & Heller, 1977) paid particular attention to psychological issues during development, recent research into software engineering seems overly focused on organizational, managerial, and work-centric concerns. The application of social science to software engineering seems restricted to the needs of system end users, particularly in the form of usability and user-centered design initiatives (e.g. Boylan, 2004; Card, Moran, & Newell, 1983; Carroll, 1997; Mayhew, 1999; Nielsen, 1993; Norman 1988; Norman & Draper, 1986). Much less attention, however, has been given to the human presence on the flip side of software engineering processes. This is troubling, given that the population of software methodology 'users' is certainly large enough to merit such study.

There is prima facie evidence to suggest that much of the value in agile and teamwork initiatives stems from the fact that they satisfy basic socio-psychological needs, and instigate important socio-psychological phenomena; above and beyond those encompassed by traditional management, organizational, and software engineering research. From a basic perceptual level, for example, we can see how rapid feedback cycles in agile methodologies turn what in traditional methodologies is an overwhelmingly invisible and long-term process into a series of tangible, visible, and confirmatory interactions. Thus agile practices seem uniquely suited to helping people deal with the complexity and uncertainty of software development environments. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

interactive and feedback-intensive nature of agile software development environment additionally corresponds with 'flow' (Csikszentmihalyi, 2000), a term used to describe the "subjective experience of engaging just-manageable challenges by tackling a series of goals, continuously processing feedback about progress, and adjusting action based on this feedback" (Nakamura & Csikszentmihalyi, 2002, p. 89).

Small group research also indicates interesting social phenomena surrounding agile teams. A number of studies posit the benefits of shared work rooms, for example, (e.g. Mark, 2002; Teasley, Covi, Krishnan, & Olson 1996), with Teasley et al. finding that productivity doubled in shared work team environments as opposed to work in cubicles. Teasley et al. found that workers who had experienced a "radically collocated team environment" showed a preference for working in similar environments again. A lack of discussion as to *why* individuals showed this preference, however, reflects a general lack of understanding regarding such phenomena in design teams.

Sutcliffe (2005) points out a gap in our understanding of design tasks with regards to human processes, suggesting the usefulness of social and psychological theory in gaining a better understanding of socio-technical systems. In particular, he offers the theory of 'small groups as complex systems' (Arrow, McGrath, & Berdahl, 2000) as useful in framing study of the patterns of communication between group members. Erickson and Kellogg (2000) further underline the importance of social awareness through the conception of "socially translucent systems," which involve the characteristics of visibility, awareness, and accountability. They note the significance of the fact that physical communication takes place in a social context, with individuals awash in ever-present social information that continuously influences numerous daily

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

actions and decisions. Finally, the recent focus on teamwork and rapid development in the competitive business context of software development makes it especially important to consider the 'psychology of software engineering,' which includes the study of social processes during product development (Sutcliffe, 2005).

There has been little research into the subjective experience of the individual in 'teamworking' (Knights & McCabe, 2000), and employee perspectives on new management techniques have gone largely unrecorded (Harrison & Storey, 1996). Mueller, Proctor, and Buchanan (2000) note that "research on attitudes towards teamworking needs to be much more critical and probing to unearth employees' 'real' rationales for their responses and behaviour, " and point out that qualitative case studies can, and have (e.g. Knights & McCabe, 2000), been used to "reveal a nuanced and variegated picture as far as employees' experience is concerned (p. 1408)." Mueller et al. further note that the value of gathering individual perspectives with regard to teamwork can be equated to the value of gathering data from individuals working in a team in addition to managerial accounts of reality.

Research into self-managing 'work teams' in organizational psychology were also unhelpful in attempts to explore positive psychological phenomena occurring in agile software development teams. A state of the art review by van Mierlo, Rutte, Kompier, and Doorewaard (2005), for example, found a "striking" diversity of outcome variables and varied results across studies on the effects of self-managing teamwork on psychological well-being. Mierlo et al. attribute the deficiency of conclusive studies to the multilevel character of the research arena, and lack of distinction between levels of theory, measurement, and analysis in most psychological studies.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The lack of empirical evidence or integrative theoretical models with regard to teamwork may also be related to the general confusion regarding the definition of teams or teamworking. General definitions can be so broad as to be essentially unhelpful; such as the definition of a team as "an interdependent collection of individuals who share responsibility for a specific outcomes of their organization" (Sundstrom, DeMuese, & Futrell, 1990, p. 20), while more detailed definitions vary greatly depending on the organizational context (Dunphy & Bryant, 1996). There seems to be value, therefore, in exploring particular forms of teamworking induced by different work environments (Scarbrough & Kinnie, 2003), such as agile teamwork within the software development context.

While agile methodologies can be associated with many interesting socio-psychological effects, they are new and complex phenomenon that have not been the subject of much scientific research (Salo & Abrahamsson, 2004). The inability of traditional cause and effect paradigms to encompass essential socio-psychological issues may be contributing to the dearth of theoretical exploration in both software engineering (see Sutcliffe, 2005; Warr & O'Neill, 2005), and teamwork (see Opie, 2000; Cooney, 2004). It was seen as valuable, therefore, to conduct exploratory qualitative research into the subjective experience of working in an agile software development team.

Research Question

The basic question explored in this study was: *What is the experience of being in an agile software development team?* To focus on the unique communicative and collaborative value added by agile practices (as opposed to the general experience of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

working in a small group environment), the research question was structured further to ask:

- How do agile practices structure and mediate the experience of individuals developing software? In particular, how do agile practices mediate the interaction between individuals and the team as a whole?
- What are the socio-psychological characteristics surrounding agile practices and individuals within a software development team? In particular, what agile practices contribute to positive feelings such as motivation and excitement during software development?

The aim of this study was to gain insight into how agile practices impact individuals within an agile team, and so be able to:

- Discover the salient aspects of agile software development and group processes worthy of future psychological study
- Gain an understanding of the context of interaction provided by agile methodologies, and so be able to better apply general (unmediated/face-to-face) psychological research to the technologically-mediated domain of software development, and
- Gain an understanding of the socio-psychological phenomena surrounding agile practices in small face-to-face teams, and so be able to instantiate such phenomena other contexts, such as in large or geographically distributed teams, or in projects ill-suited to agile methods.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Research Context

The socio-technical view, as discussed in the next section, highlights the importance of exploring the domain and context of phenomena under study. Badham, Clegg, & Wall (2000) note that "the optimum socio-technical design is contingent on the purposes and environment of that [system]." This inextricable link between today's tasks and technology (Coakes, 2000) makes it therefore essential to gain an understanding of the unique characteristics of the software development context.

Characteristics of Software Systems

A common metaphor in explaining the characteristics of software systems is to compare software development to construction. Civil engineers in construction work with physical materials subject to the laws of physics. These materials must be tailor-made, transported on-site, and constructed with great effort to form, for example, a bridge. In comparison, software engineers work in a virtual programming environment where the laws of physics do not apply. Construction or reconstruction of software involves mammoth effort, not in lifting stone blocks, but in logically connecting lines of code to form a software product.

Brooks (1987), in his influential essay *No Silver Bullet: Essence and Accidents of Software Engineering*, outlines the defining characteristics of software systems. He states that the difference between dealing with complex physics and complex software systems is that physicists can be pretty well assured that the objects of their study can be simplified down to unifying principles of physics. In contrast, software engineers are dealing with *man-made software artifacts* created in an overwhelmingly ad hoc manner;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with each artifact being shaped by an amalgamation of techniques, structures, and tools, which in turn stem from the various people and environmental contexts present at the time of development.

Brooks goes on to note that new software systems are necessarily defined by *conformity* to previous complex and disordered systems. This results in additional complexity when creating software to be compatible with existing technology. Imagine for example, a team required to design an add-on for an existing software product. This requires that they understand and work with the previously built software system; a system developed for a different purpose, by a different team, and perhaps a very long time ago using different processes and tools. Especially unhelpful is the fact that the success of a software system can be judged primarily on whether it does what it is supposed to do, rather than how it does it. Consequently, the internal workings of a system are not necessarily well written, logical, or well documented. This makes it especially challenging to develop new software that must interact closely with the old.

Brooks notes the *lack of physical constraints on software complexity*, leading to infinitely malleable components and systems. Physical systems, such as cars and bridges, are limited by the materials available in nature. Software objects, on the other hand, are made up of systems of components whose properties and interactions have been explicitly defined by programmers as needed. He additionally offers *uncertainty* as a corollary to complexity. Heterogeneous elements within software systems interact in a non-linear fashion, which creates effectively unpredictable results in system behavior. So while a civil engineer designs a bridge using well-tested mathematical formulas regarding

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the nature of physical objects, a software developer has no such tried and true formulas on which to rely.

Another unique property of software offered by Brooks is the *relative ease of structural change* at any point in development. This *flexibility* can again be held in contrast to physical engineering disciplines. Once construction on a house has started, for example, the costs of change are exponentially larger than before the foundations have been set, whereas lines of code in a software program can be changed by a few keyboard strokes once a change has been defined. Brooks also attaches the property of *invisibility* to software systems. Hence, unlike architects, geographers, or mechanics, who can map their domain knowledge onto geometric drawings of components and physical interactions, software engineers suffer from trying to map non-locative, non-physical entities and interactions onto shapes or text. The user interaction of clicking through dialogue boxes and entering data, for example, occurs over time, and becomes increasingly complicated as user choices branch out, and as system responses to each choice must be defined. Brooks holds that software structures are "inherently unvisualizable," making it difficult to model and represent software in a way that will assist in understanding and decision-making.

It is worth pointing out that many system interactions are not only unvisualizable, but impossible to predict before the system has been developed. This is especially true for user interaction and in-context use, as end-users generally use systems in ways unforeseen by initial developers. Also worth noting is the fact that not every part of a software system is invisible or infinitely changeable. The user interface component of software systems necessarily has a visual representation, and does not fully retain the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

properties of changeability or invisibility existing in the 'backend' (i.e. the part of the system that end-users cannot see). This has implications for the development of the user-interface in software systems, and presents some issues in the practice of agile methodologies (Patton, 2002).

Overall, we can see fundamental characteristics of software such as complexity, uncertainty, flexibility, and invisibility that affect the task of software development, and can be held in contrast to traditional engineering practice.

Characteristics of Software Development

Software development can be seen primarily as a *design* activity (Fowler, 2005; Reeves, 1992). Good software design involves the production of good code, which is defined and refined by tasks such as architectural design and software testing (Reeves, 1992). Development is influenced by the fact that software code is easy to generate; a simple software program can be constructed in a few hours, for example, while a more complex module can be created in a couple of days.

Fowler (2005) notes how "most software development is a chaotic activity," with software being created out of a series of short-term decisions without much of an overall plan. This approach encounters problems, however, as systems increase in size. Commercial software products, for example, involve hundreds of thousands of lines of code, with many running into the millions. For this reason, team coordination in software development is paramount, particularly since choices made at each level of software design will likely affect other levels of the software system (Reeves, 1992).

It follows that software development is much more than just programming and code. The creation of software involves a large number of stakeholders, including but not

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

limited to, developers, business analysts, usability engineers, sales and marketing, quality assurance, sponsors, and customers, each of whom may be part of a cross-functional development team, or may simply provide input into the project as needed. Thus individuals with specialized skills and roles must coordinate their efforts to create quality software, and a single product must resolve diverse and often conflicting requirements from numerous stakeholders. Resource constraints on time, budget, and available technology and skills must also be accounted for.

The involvement of customers, who often exist apart from the organization developing the software, is particularly significant. The customer is the one for whom the software is being made, and who must ultimately be satisfied with the final product (Turner & Boehm, 2003). The need to involve and satisfy the customer, along with the importance of close team coordination, make estimation and scheduling activities essential to software development. An important distinction to make at this point is the difference between customers and users. The term customer, as generally used in software development, indicates the business customer; i.e. the person(s) who requests and pays for the software to be developed. The customer is the one who understands and determines the business value of a software system and its constituent parts. End-users, on the other hand, are the people who use the system at the end of the development process. This distinction is a source of some confusion, particularly in agile methodologies (Fraser et al., 2004; Martin, Biddle, & Noble, 2004).

Problems in software development are generally one-off or unique (Highsmith, 2004), with each new project involving a different set of requirements and constraints. In addition, software design is open, meaning that there are many possible solutions that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

could be implemented to satisfy project constraints. Once created, however, a system will often persist, and needs to evolve or adapt to suit new needs or contexts. This evolution is particularly important in the 'new' or 'information' economy. Greater dynamism and uncertainty in this new business context have increased the importance of adaptability and flexibility, as well as the ability to make decisions quickly and in the absence of complete information (Merali, 2004). The need for flexibility has consequences both on the programming level, where software must be developed and managed to allow for future change, and the project level, where new and changing requirements must be permitted and managed throughout the course of a project (Beck, 2004).

Another characteristic of software development is that it has been a practice plagued by failure. The 2004 edition of the well-known CHAOS report (The Standish Group, 2004) on project failures in software development found that only 34% of software projects are considered successful, while 15% are declared failures, and the remaining 51% can be grouped under the label 'challenged.' While these results are a dramatic improvement on the dismal statistics reported in the original CHAOS report (The Standish Group, 1994), they are still less than impressive.

The failure statistics are especially regrettable when we consider the fact that, by and large, software developers are passionate and proud of what they do. In *The Soul of a New Machine*, for example Tracey Kidder (1981; see also Coupland, 1996) describes the hope and anxiety that accompanies the development of a software product, and the dedication of developers in projects plagued by obstacles. Though fictional, the book is based on, and reflective of the experience of software developers working in industry. It

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

seems that for a long time software developers have been dealing with frustrating working conditions, including impractical software requirements, unrealistic deadlines, responsibility without authority, and insufficient resources, such that "doomed projects" are an accepted part of software engineering culture (e.g. Yourdon, 2003).

Summary of the Domain of Software Development

Software development is a relatively new activity, which we have been engaging in for just over 50 years (Pressman, 2005). As such, the discipline is dealing with growing pains (Boehm, 2002; McConnell, 1999). Modern software design is vastly more complex than it was just a few years ago, with every indication that software will continue to become increasingly sophisticated. Increased business demands have compounded this issue, and software development has rapidly evolved from a domain in which individual programmers could feasibly create workable products, into one that requires increasingly large-scale and team-centered activities.

Software development involves layers of complexity: in creating system code; in incorporating code into an integrated software product; in coordinating teamwork throughout software development; and in resolving differences in requirements coming from diverse stakeholder groups. It can be held apart from the problem solving traditionally studied in psychology, where a well-defined problem with specific solutions is solved, often in a laboratory environment (Curtis, 1984). Rather, software development involves *unstructured and complex problem solving* in a semantically rich domain and environment.

Software development is also an *intensely social activity* (discussed in Warr & O'Neill, 2005), involving people and relationships between people. Software creation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

involves more than the management of the software artifact, just as knowledge sharing involves more than just the management of information (see Wenger, McDermott, & Snyder, 2002). Team members must collaborate to optimize a product, while at the same time considering the constraints and opportunities of personal, project, organizational, and business contexts. A fitting description is provided by Alistair Cockburn (2002), who defines software development as a "goal based game of invention and communication."

A paradox lies in the nature of software development as a problem solving activity. The unstructured nature of the software problem makes informal interaction the best means for communication and collaboration (Kraut & Streeter, 1995). At the same time, informal communication seems insufficient in the face of such a complex activity; one which requires team coordination, and the management of many small details and interdependencies that can easily overcome the strict limitations of the human brain (discussed in Sutcliffe, 2005). This paradox leads to inherent difficulties in the development of methodologies and practices to manage software development activity, and often strain on the people who carry them out. Brian Marick notes (Mudgridge & Cunningham, 2005, p. xxi):

> "A software project is a place where different cultures come together. Some people face towards the business and its concerns; other people face towards the computer and its demands. To an expert in financial trading, a "bond" is something that's tangled up in all sorts of explicit and implicit legal, social, historical, and emotional meanings. To programmers, a Bond is an object in their program that they are trying to *keep* from getting too tangled up with other objects, lest their brains explode. Somehow, these people have to work

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

together..."

While the 'software crisis' of the 90's (Gibbs, 1994; The Standish Group, 1994) seems to have lessened, software engineering is a discipline still trying to come to terms with the demands placed on it by modern society (Glass, 2002). Solutions to date have included numerous tools and processes to aid in system design, management, and construction. The structured design paradigm that has dominated software development methodologies for the past 30 years is based on the 'waterfall' model. This model was originally described by Royce (1970), and instituted as the development strategy of choice in government organizations such as the US Department of Defense (1988, 1994, 1995).

Waterfall methodologies, inspired by practice in engineering disciplines, coordinate software development projects to progress through sequential phases. These phases generally include: *planning* and estimation of project goals; *analysis* and modeling of system requirements; detailed *design* of software based on analysis; *implementation*, or coding of the product itself; *integration and testing* of the system in a testing environment; *installation* or deployment of the software, often to the user base; and finally *maintenance* of software system over time. The purpose of such a sequence is to make software development more predictable and efficient, and so avoid costly errors.

A common argument against the waterfall method is that it is overly prescriptive, with the long initial periods of planning, analysis, and design ill suited to the nature of software and software development (e.g. McConnell, 2004). Further criticisms of waterfall-like methods are that preoccupation with thorough documentation detracts

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

from prompt delivery of a final product, and increased compartmentalization of team roles impedes innovation and integration efforts. Also striking about this type of methodology is the overwhelming focus on the software artifact as opposed to the people creating it.

Agile Software Development

The following summary of agile is not meant to be exhaustive, with such analysis being found elsewhere (e.g. Abrahamsson, Salo, Ronkainen, & Warsta, 2002; Hayes & Andrews, 2006). The exact definition of agile is elusive, and it is important to note that the term encompasses several different methodologies, including Feature Driven Development (FDD) (Coad, deLuca, & Lefebvre, 1999), Dynamic Systems Development Method (DSDM) (Stapleton, 1997), Crystal (Cockburn, 2005), Agile modeling (Ambler, 2002), Scrum (Schwaber & Beedle, 2002), Adaptive Software Development (Highsmith, 1999), and Extreme Programming (XP) (Auer & Miller, 2001; Beck, 2000, 2004; Jeffries, 2001). The description below focuses on practices associated with Extreme Programming, the most often cited and well-known agile methodology, with reference to other agile literature as found to be relevant to this study.

Building Software

The XP process generally starts with *customer stories*. Customers are instructed to come up with a list of well-defined features or stories that they would like included in the software. This allows *the planning game*, where the development team estimates the time and resources needed to implement each story, and customers use these estimations to select stories they want included in the current iteration (version) of the software based on prioritized business value. Developers begin software production before designing a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

complete solution, and software is developed in *short iterations*, generally 1-4 weeks in length, at the end of which functional software is completed and tested.

The practice of *self-organizing teams* (Cockburn & Highsmith, 2001; Lindvall et al., 2002) is common in agile. Rigid team structure is avoided, with programmers often able to choose the stories that they would like to work on for the current iteration. Such teams organize themselves in a way that best completes the project at hand, given available resources and the particular capabilities and competencies of the individuals in the team. Full *releases occur often*, generally every 4-6 months, at which point a version of the software can be made available to customers. Agile practices allow *early, concrete, and continuous feedback*, between customer and developers, as well as between the team and the product that is being built. This enables continual negotiation and renegotiation regarding the nature and scope of the software, and how it is to be developed.

An essential XP practice is *test-driven development* (TDD) or *unit testing* (Ambler, 2003, Ch 11; Beck, 2003, 2004; Marick, 2006). TDD is a practice by which code is added in small increments to satisfy simple functional requirements. Before writing any program code, developers write 'tests' that the code must pass. For example, when writing code to support user log in, the tests would concern possibilities in system action, such as what must occur when the user presses enter before entering anything in the log in box, or what will happen when they enter the wrong password. Developers write tests that verify the correct system response, that is, the tests will pass if the system responds correctly. They first make sure that the tests fail (since the functionality doesn't exist yet), then write only the code necessary to make the test pass.

Test-driven development serves a number of purposes. Firstly, the tests provide a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

clear focus for developers as they code, with tests and accompanying code often written on a minute-by-minute basis. They also provide feedback on the validity of code once written, similar to the way usability benchmarks are used in interface design. Much of the value of testing comes from the fact that developers accumulate a growing set of *'regression' tests* that can be quickly run to make sure that new functionality does not 'break' any previously written code, i.e. cause it to stop working. Automated tests are used to allow rapid and dynamic interaction and feedback on progress. Tests also provide a means of documentation and communication (Ambler, 2005) in that they effectively summarize the purpose and function of each system module. In addition to programmer tests, there are also *customer acceptance tests.* These are automated tests defined by the customer regarding desired functionality, and are run to ensure that features are implemented correctly (Jeffries, 2001).

Agile processes place value in *simple design*, with a focus on constructing a working product based on the customer stories for the current iteration only. An agile catch phrase declaring "You Ain't Gonna To Need It" (YAGNI) is used to prevent inclusion by anticipation. This puts a focus on building software for today, rather than preparing for possibilities in future iterations. Value is also placed on a *working version of the product,* maintained on a weekly or daily basis. *Incremental, or evolutionary design* (Fowler, 2004) is also practiced. Rather than spending a lot of time planning, analyzing, and designing for a big perfect release at the end of a long development cycle, agile developers do each of these activities, a little at a time, as development progresses (Beck, 2000).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Time-boxing is another common practice used to assist simple design and scheduling. Development activities or projects are split into separate time periods, with a set number of hours allowed for each task. For each period, deadlines and resources are fixed, while deliverables are considered more flexible. Thus the *scope* of a development effort is adjusted to meet scheduling constraints, and functionality that will not fit into the current time period is dropped or reconsidered, usually in negotiation with the customer. At the end of the time box each task should be one hundred percent completed.

In place of defining the finished product at the outset of the project, XP developers utilize *continuous integration and refactoring*. As each programmer completes his or her tasks, they continuously integrate their individual code into the collective system code, often on an hourly basis. Tests are utilized at this point, where a combined suite of unit tests from the entire team must run one hundred percent correctly each time code is integrated into the system. This avoids problems often encountered in non-agile methodologies, where pieces of code can exist and be developed separately for days, weeks, or even months, with disastrous results when attempts are made to integrate them into a single workable 'build' (version of the software system).

Refactoring (Fowler, Beck, Brant, Opdyke, & Roberts, 1999) involves improving the structure of code for simplicity and understandability (and therefore flexibility and changeability), while retaining the same functionality. Basically, software developers review program code to ensure that it is the simplest possible design for the given functionality. If not, they refactor, or revise the code so that it is. This avoids the problems of 'spaghetti code,' which is so complex and convoluted as to be indecipherable.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Dependent on code simplicity and readability is the practice of *collective code ownership.* No one person in an XP team 'owns' any part of the code, and any pair of programmers can change code anywhere in the system as they see fit. This removes potential bottlenecks and complications in the coding process. In support of this, agile teams often stick to a *coding standard,* whereby all code in the system is written in a similar manner.

Finally, as mentioned with regards to unit testing, *automation* is extremely important when implementing agile practices (Ambler, 2005). Automated tests and software builds make it quick and easy for team members to see how their code works, both alone and when integrated into the main software system. The ease and speed provided by automation is especially important, since lengthy manual processes are likely to be skipped in the face of time constraints (Jeffries, 2001).

Communication and Collaboration

Agile methodologies put a strong emphasis on constant communication and coordination between team members, particularly face-to-face interaction. *Rapid feedback* is highly valued in the agile communication structure, and lines of communication are kept open and short. If there is any question regarding an aspect of software design that a particular developer is working on, he or she will initiate a short (15 minute) meeting with the person most likely to have the answers that they need. *Stand-up meetings* are a common practice, involving 10-15 minute meetings at the beginning or end of every work day, with each person in the team giving a brief run down on their progress and/or plans for the day.

The customer driven nature of agile is one of the main factors setting it apart from

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

traditional methods (Turner & Boehm, 2003). Agile stresses the importance of a dedicated *on-site customer* representative who "provides the requirements, sets the priorities, and steers the project" (Jeffries, 2001). This can be held in contrast to traditional methods, where interaction with the customer is relatively limited, usually occurring during initial planning and specification phases. The agile customer has much more control over the software development process. Working software is delivered regularly, allowing them to see the product, evaluate the software themselves or with end users, and provide feedback. The quick feedback between the customer and the development team is viewed as a critical success factor in agile projects (Lindvall, 2002).

Agile teams operate in a *shared workspace* or "war room," which basically consists of a large open plan room, with workstations set up so that developers can work in a shared environment. Small cubicles are made available in case team members need privacy to work, but the bulk of activity occurs in the common space. War rooms facilitate communication and rapid feedback between team members, and have been found to be astoundingly effective in increasing team productivity (Teasley et al., 1996).

Agile teams also operate in *informative workspaces*, which generally involve populating the working environment with *information radiators* to effectively disseminate information between team members (Cockburn, 2002). This could include putting up posters outlining end-user personas, charts specifying project schedules, or elaborate post it note constructions outlining which customer stories have or have not been completed, and by whom. Information radiators are posted on common wall space so that everyone can see them, and are often marked up or changed as the project evolves. The goal of these artifacts is to ensure that everyone is on the same page in their

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

understanding of the project goals, and can quickly see what state the project is in at any point in development (Beck, 2000).

Agile teams also tend to put a lot of emphasis on *interactive communication artifacts* (e.g. Bellin & Suchman Simone, 1997), where discussions and meetings occur around the use of flip charts, white boards, index cards, and post it notes. *Software and code* are also utilized, with a working version of the software acting as a shared artifact around which activity and communication is centered. *Unit tests* can further be used as both requirements specifications before code is written, and documentation of what the code entails after the tests have been satisfied.

Tools created to support agile software development have become increasingly important to team efforts. Tools such as JUnit (Husted and Massol, 2003; Object Mentor, 2006) and Nunit (Hamilton, 2004; Two, Poole, Cansdale, & Feldman, 2005) provide *automated frameworks for writing and managing tests*. Such tools automate what would otherwise be tedious manual processes, and give immediate visual feedback on whether the tests have passed or failed. Testing frameworks such as Framework for Integrated Tests (Fit) (Mugridge & Cunningham, 2005; Shore, 2005) and Fitnesse were developed specifically to enhance communication and collaboration with customers.

Using these frameworks, customers (or users) themselves can write acceptance tests into simple, spreadsheet style documents. These tests are then checked against the actual code, allowing the customer, without any knowledge of programming, to interface with the software product directly and in real time. The increased interaction between developers, customers, and code permitted by such tools greatly enhances feedback, code development, and understanding of the software problem on all sides. The resulting

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

records of tests can further be used for documentation purposes.

Finally, we come to *pair programming;* a practice that has garnered much interest in the software engineering discipline, being perhaps the most well examined agile practice to date (Nicolescu & Plummer, 2003; McDowell, Werner, Bullock, & Fernald, 2003; Chong et al., 2005; Hanks, 2003; Hanks, McDowell, Draper, & Krnjajic, 2004; Nosek, 1998). In pair programming developers work in pairs for any coding activity. Pairs usually sit at the same workstation, with one person 'driving' at the keyboard and the other sitting beside them 'navigating' and providing input. A common practice is for programmers to switch pairs regularly, such as every 90 minutes, or whenever the current task is completed. Along with an extra person considering and reviewing each coding activity, pair programming offers benefits such as an apprenticeship-like learning environment, and dissemination of domain knowledge, project understanding, and coding skills across the programming team.

On the whole, knowledge dissemination, communication, and collaboration are enhanced through the use of various agile practices and tools that maintain wide and open channels of interaction.

Agile Interactions

A number of themes and ideals permeate agile interactions, helping smooth coordination of the team and development process. *Feedback* has been mentioned as important in both the physical construction of software, and in communication and coordination between the development team. Another role that feedback plays is in *team reflection or retrospectives* (Kerth, 2001). While less common than many of the other practices, agile teams are encouraged to take time at the end of a project or iteration to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

consider their processes and team interactions and reflect on how they could improve or adapt.

Also mentioned is the idea of *simplicity,* summed up in the agile catchphrase 'Do The Simplest Thing That Could Possibly Work' (DTSTTCPW). This principle is applied rigorously to every aspect of the development process, and is used to guard against the very human tendency to put large amounts of time and effort into getting something 'right', while adding little value to the desired end goal. This is related to the previously mentioned YAGNI ('You Ain't Gonna Need It'), which guards against the tendency to over prepare for later contingencies that may never actually arise. The main example of this principle can be seen in the use of documentation. In contrast to traditional software development processes, which focus much time and effort into creating and maintaining detailed documentation, agile documentation is maintained only to the extent that it helps the immediate goal of creating software. Note that programmer documentation involves work internal to the software development processes, and should not be confused with user documentation such as usage manuals or help. Agile software development principles, such as DTSTTCPW, focus on simplicity and short-term value to allow for change in the future, and are specifically tailored towards a business context where future requirements are unknown (Boehm, 2002).

Sustainable pace is an XP practice based on the idea that people can work better and longer when they have personal and down time. As such, agile team members are not encouraged towards, and are even discouraged from, working overtime. This is related to a general aversion to unrealistic deadlines and the stress that accompanies it. Finally, agile teams maintain a *whole team mentality,* and a level of *shared leadership,*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where input from team members is encouraged and valued, and each team member has a say in the direction the project is going.

Martin Fowler (2005) points out two things that define agile methods. Firstly, agile methods are adaptive rather than predictive. They are designed to deal with change, as opposed to trying to plan large chunks of development under the assumption of no change. Secondly, agile methods are people, rather than process-oriented. The goal of agile is to define a process that will work well for the team and the project at hand, rather than adhere to a fixed process regardless of other factors.

There are additional aspects of XP and agile software development that deserve mention. Firstly, many agile processes, and XP in particular, are highly integrative methodologies. In other words, specific practices in XP depend on other practices in order for them to be effective. For example, the ability to respond to changes in customer requirements from one iteration to the next is highly dependent on the code simplicity due to refactoring and coding standards. The specific dependencies in the XP process are further outlined by Beck (2000, 2004), but it is worth noting that many agile practices are not standalone procedures.

Also important are the culture and values surrounding the methodologies. Kent Beck (2004) offers the underlying values of courage and respect in addition to communication, simplicity, and feedback, while Cockburn and Highsmith (2001, p.132) state that "agility requires that teams have a common focus, mutual trust, and respect." An increasingly complex set of values comprise agile ideology and culture, and feed into the experience of individuals interacting within agile environments.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Conceptual Framework

Agile methodologies prescribe recursive human action and interaction in a way heavily mediated by tools and processes. This led to the definition of agile teams as complex adaptive socio-technical systems. This definition follows growing trends in research and theory from information systems (IS), computer supported cooperative work (CSCW), and operations management, and is in line with the common conceptualization in psychology of work teams as complex adaptive systems (McGrath, Arrow, & Berdahl, 2000; Ilgen et al., 2005). IS research was found to be particularly relevant in framing this study, with software development methodologies fitting the definition of information systems as "systems comprised of people, machines, and methods organized to collect, transmit, and disseminate information" (ATIS, 2001).

The Value of Socio-technical Systems

The value of a socio-technical system framework can be summed up in the following quote from Harrison and Story (1996, p. 63):

> "On the one hand, the operations management literature tends either to ignore the social and organizational dimension or pays scant attention to its importance. On the other hand, the organizational behaviour literature has not realized its potential because of its failure to engage fully with the technical and operational arrangements which it wishes to critique. In consequence, both sets of literature, while ostensibly assessing the same phenomenon, end up talking past each other, and their treatment of the recent important developments in manufacturing methods remain partial."

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Coakes (2000) describes socio-technical system research as "the study of the relationships and inter-relationships between the social and technical parts of any system." Such research addresses the fact that human and technical subsystems interact and mutually adjust, often with dramatic effect. Socio-technical systems thinking is strongly influenced by von Bertalanffy's writings on open systems and general systems theory (1950, 1968) involving principles of feedback, emergence, and a whole that is more than the sum of its parts, as will be discussed later.

In operations management and work design, the socio-technical view is held in opposition to Taylorism (see Taylor, 1947), which breaks each job down into its elements and reorganizes them into the most efficient task-oriented combination, such as on a car assembly line. Taylorism holds the requirements of the machine above social, or human requirements, and it can be argued that this has lead to a workforce that is bored, dissatisfied, and relatively inefficient.

The best-known articulation of socio-technical system theory comes out of the work of researchers from the Tavistock Institute of Human Relations. Particularly, Trist and Bamforth's (1951) research into the introduction of longwall mining methods to the Durham coalmines in England. Their paper *Some Social and Psychological Consequences of the Longwall Method of Coal-getting*, outlines the social shortcomings of the newly mechanized procedure, and describes the dysfunctional working environment that resulted from the disruption of informal social structures in the mine under study. It was found, for example, that the longwall mining procedure had process interdependencies that made it especially vulnerable to local disturbances; hold ups at one

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

point in the process would therefore resonate about a relatively large social space, creating norms of low productivity and decreased worker morale.

Trist and Bamforth also found naturally occurring differences in worker organization, with corresponding effects on worker satisfaction and commitment. They identified different types of groups within the mining community: 'conventional' groups who organized themselves according to task; and 'composite' groups who maintained complex work roles and cooperation in achieving whole-team goals. It was found that the latter formation resulted in significantly higher productivity and lower absenteeism than the former. The longwall study, in addition to identifying the organizational phenomena of self-regulating (composite) work teams, brought to light the fact that different social arrangements can allow or undermine the success of technology. Its interesting to note that while socio-technical theory stemming from the Tavistock research was strongly oriented towards worker rights and well-being, modern incarnations of socio-technical theory, as found in the field of information systems for example, often seem to omit these considerations.

The socio-technical systems view is notable in that it avoids social determinism, as well as technological determinism. The reduction of technologically mediated interactions to purely social forces ignores the power of technology to indelibly instantiate certain patterns of human action and interaction (Friedman & Nissenbaum, 1996). In discussing the role of technological artifacts on politics, Winner (1986) describes how technology can make possible or impossible certain actions, and so embody values and power. He gives the example of low-hanging overpasses present along the parkways of Long Island, New York. While seemingly innocuous in design, the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

overpasses meant that racial minorities and low-income groups, who traveled on buses too tall to drive under the bridges, were restricted from reaching certain affluent areas of the city. Technology is a force that can mandate or restrict, as well as amplify or mute human activity. In this way technology makes some actions easy to do, and some actions hard to do, often with wide-ranging consequences on both individuals and society. As such we should be cognizant of its presence and effect. This is particularly important when we consider that it is we who define and create the technological constructs that make up our environment (Lessig, 1999).

The role of technology in contemporary team interactions has been seriously understated, even while "positing a motivational basis for the evolution of team-work designs will only take us so far in understanding the causes and consequences of the changes currently taking place" (Cooney, 2004, p. 12 citing Cordery, 2003; Jackson, Wall, Martin, & Davids, 1993). This does not mean, however, that we should swing our focus onto technology to the exclusion of human issues. Catering to ingrained cognitive processes and the limitations of human thought is essential to the effectiveness of software development efforts (Sutcliffe, 2005; Curtis, Krasner, & Iscoe, 1988) and the well-being of those involved in development.

The dichotomy between technology and humanity is firmly entrenched in the field of software engineering. Software design processes coming from the realm of engineering and computer science commonly ignore human issues to focus on technological constraints and opportunities. Socio-technical theory suggests that a good fit between tasks, people, and technology will increase effectiveness, allowing systems to become more integrated over time (Arrow et al., 2000). Such joint optimization will

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

benefit, not only the users whose social needs will be met, but the system as a whole (Mumford, 1995).

Agile methodologies are made up of an amalgamation of values, principles, and practices spanning human, as well as technical issues in communication and collaboration leading to software development therefore seems fitting. More than the business perspective that project management places overtop of traditional methodologies, the tools and practices in agile seem firmly based on basic social and psychological principles that help make the processes of developing software more 'user-friendly.'

Systems Thinking

Systems thinking is a determining influence in both socio-technical systems theory and small group interaction theory, and a framework already used in some instantiations of agile software development (e.g. Highsmith, 1999). The following sections will provide a brief overview of systems thinking, and discuss how aspects of systems theory were used to frame this study.

A Brief Introduction to General Systems Theory

General systems theory, originally developed by biologist von Bertalaffany (1968), is a useful conceptual tool in describing and modeling interactions in complex environments. Since its inception, systems theory has been adopted by thinkers in such diverse fields as mathematics, linguistics, economics, management, sociology, and psychoanalysis, and has accumulated an accepted set of universal characteristics found in all naturally occurring systems, or what Goodman (1982) describes as the logical outcome of changing environmental conditions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An essential contribution of general systems theory is the conceptualization of *a system as a holistic entity* - a unified whole consisting of more than the sum of its parts. Systems are composed of *interdependent and connected parts* (Badham et al., 2000) that mutually interact as part of a *purposeful whole*. Rather than reduce an entity into parts or elements, systems theory explores the arrangement of elements within a system and the interactions between them. *Emergent or synergistic properties* arise in systems through component interactions, comprising characteristics that cannot be reduced to the addition of the properties of system parts or components. The emergent property of personality, for example, can be seen to arise from the actions and interactions of neurons in the human brain.

Within systems, a large number of quantitatively small 'micro' actions can result in qualitative 'macro' differences in the system as a whole. Emergence becomes increasingly hard to predict as systems become more complex, with the number of interactions between components increasing exponentially with the number of components. This does not mean, however, that a large number of components in a system will necessarily result in emergent properties. Interactions between components may, for example, not reach critical mass, may cancel each other out, or may simply be irrelevant to the development of such properties.

Emergent properties result (or fail to result) from another important aspect of systems theory, namely the *dynamics of regulation*, which include feedback, self-organization, and recursive causality (Leonard, 1994). *Positive and negative feedback loops* exist within systems, respectively enhancing or reducing component activity, and allowing snowball or spiral effects. *Self-reference and circular causality* through

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

feedback and feedforward processes creates outcomes that often defy predictions, with each micro level activity being contingent on the state of the system, and having a resulting effect on the system in turn. At the same time, there seem to be universal laws that govern the evolution and form of systems. Systems are likely to be *self-organized*, with system dynamics tending towards increasing the inherent order of a system.

Systems theory recognizes that the best outcome for a given system results from the *coordination of subsystem activity*, as opposed to local optimization of subsystem outcomes (Leonard, 1994). Cause and effect relationships studied in the reductionist tradition are unlikely to hold in a system environment where the components recursively interact. Isolating parts of a system for further study will result in a less than complete understanding of a system, and so a system is viewed as a dynamic whole, with attention given to interactions, feedback, and controls.

The Agile System

Merali (2004), in a discussion of complexity and information systems, notes that the Internet and its attendant technologies have contributed to a sharp rise in the level of complexity experienced in the developed world. She explains how technological developments have allowed increases in connectivity, in storage and processing capacity, and in the reach, range, and rate of information transmission. Merali goes on to describe the 'new' economy, which includes increased dynamism and uncertainty in the competitive context, and the need for quick decision making in the absence of complete information. Flexibility and adaptability, along with information and knowledge, are of utmost importance in maintaining competitive advantage. She calls into question the adequacy, in such a context, the "formalisms of decision making" and classical paradigms

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of science and design, and talks of network forms of organization which resolve and exploit the complexities in today's world.

Agile is one such form of dynamic and networked organization, with agile teams instantiating themselves as complex adaptive systems in order to create complex software systems. We can see general systems principles reflected in agile practices. For example, the agile focus on collective (holistic) action, feedback and feedforward mechanisms, and interconnections between team-members.

The first step in classical systems studies usually involves defining the boundaries of the system (Leonard, 1994). A system can be seen as composed of interconnected subsystems, which can themselves be decomposed into interconnected subsystems. Each system level, modularized in this manner, can be viewed as a black box entity whose internal complexity can be ignored (Merali, 2004). System boundaries seem to break down when systems involve adaptivity and human agents, as will be discussed later, but for now the boundaries of the agile system studied in this research will be defined.

To identify the boundaries of the system, or alternatively, define the level of analysis of human activity, I utilized Curtis et al.'s (1988) framework for behavioral analysis in software systems. The model proposes that behavioral analysis of the generation of software systems should occur at several levels, including psychological, social, and organizational levels. It highlights the fact that when a software development task exceeds the ability of a single person, development occurs in a team, causing social processes to interact with individual cognition and motivation. Higher up the model, and somewhat beyond the scope of this study, Curtis et al. describe interactions between teams, as well as organizational and business contexts.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

When applying this framework to a study of subjective experience in agile teams, the focus is on individual human systems existing within the surrounding system of a software development team. In other words, this study focuses on individuals within the environment of an agile software development team. In reference to the socio-technical focus of this study the study of relationships between individuals was amended to include examination of how technology mediates the relationships in the agile system.

Systems theory directs attention towards the relationships between system components. This avoids some of the difficulty in studying agile teams, which are by nature, adaptive and suited to their particular environment. Biologist Humberto Maturana describes systems that are autopoietic, or "self reproducing, where the structure of the system can change, but the organization remains invariant" (cited in Paetau, 1996). This study maintained a focus on identifying such invariant relationships between agile system components, regardless of the particular structural incarnation of agile employed by teams. The systems theory framework also directs attention towards the exploration of particular system properties, including: feedback and feedforward loops, where information is amplified or muted at particular system nodes; interdependencies between system components; and system goals, and how system components are aligned with these goals.

Human Agency in Systems Theory

In what has been called the "first paradigm shift" in systems thinking (Gharajedaghi, 1999), systems can be organized into three types: deterministic, or mindless systems; animated, or unminded systems; and social, or multiminded systems (Gharajedaghi, 1999, p.10-13; Ackoff, 1999, p. 27). These types of systems are based in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the physical sciences, the natural sciences, and the social sciences respectively, with the development of general systems theory being historically grounded in the first two fields of study. The introduction of human agents adds additional complexity in that each component or individual within a *social system* is purposeful, having his or her own free will or intent.

In discussing the nature of multiminded systems, Gharajedaghi notes, "behavior of a system whose parts display a choice cannot be explained by mechanical or biological models. A social system has to be understood on its own terms...the critical variable here is purpose. According to Ackoff and Emery (1972), an entity is purposeful if it can produce 1) the same outcome in different ways in the same environment and 2) different outcomes in the same or different environments. Although the ability to make a choice is necessary for purposefulness, it is not sufficient. An entity that can behave differently but produce only one outcome in all environments is goal-seeking, not purposeful. Servomechanisms are goal-seeking, but people are purposeful. As a purposeful system, an organization is part of a larger purposeful whole, the society. At the same time, it has purposeful individuals as its own members. These three levels are so interconnected that an optimal solution cannot be found at one level independent of the other two. Aligning the interests of the purposeful parts with each other and that of the whole is the main challenge of the system(p.12)."

This development of systems theory suggests a number of things to a study of the agile software development. Firstly, it outlines the importance of observing *system and subsystem goals*, and how they align with each other, as opposed to traditional cause and effect models of human behavior. The teleological view, concerning output or end state

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can be held in contrast to the deterministic view, which concerns itself with only the effect of input into the system (Ackoff, 1999). This highlights the importance of studying positive psychological phenomena, such as pride, that have been understudied in psychology (Peterson & Seligman, 2003), and have been given particularly little attention in both management theory and practice (Goshal, 2005).

Secondly, it suggests viewing the interactions between system levels. Merali (2004) notes that boundaries separating the system from its environment, which play an important role in traditional systems theory, are inherently different for social systems due to the dynamic nature of social network interactions and the permeable nature of organizational boundaries in practice. Human awareness introduces additional complexity into the study of system behavior, with human psychological and social forces crossing system boundaries. Human agents are additionally aware of social/relational information as well as task-based information, and of entities existing beyond their immediate environment, both in time and space. This is especially true in a socio-technical system, where information can be rapidly disseminated throughout the system, overcoming limitations of local optimization or bounded rationality.

The 'non-modular' nature of humanity (see Demarco & Lister, 1999) is an interesting starting point of study, since process and system models that do not take this into account work very differently in practice than in theory. Despite this, the merging of psychoanalytic and sociological understanding of groups is not common (Anderson, Carter, & Lowe, 1999). Study of system level interactions does exist, however, as in the work of psychoanalyst and group therapist Bion (1961; see also Vogler, 2000). Coming from the Tavistock Institute, Bion makes the distinction between groups as collective

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

entities and the individuals within them, stating that groups exert considerable influence on individual members. This study maintained a focus, therefore, on interactions and social forces between the individual and group levels of the agile team systems.

Related Literature

Related literature was initially explored in search of a theory or framework that could encompass research questions and domain, and later examined to help define the scope and focus of a grounded theory study of individuals in software development teams. Agile phenomena sit on the cusp of several disciplines of study, involving the experience of an individual (psychology) communicating within a group (social psychology, group psychology), practicing a certain kind of software development (systems analysis and design, software engineering), while at the same time utilizing specific communication structures and processes that are highly mediated by technology or artifacts (communication, information systems, computer-supported cooperative work, operations management). Moreover, all activity is occurring in the context of a software development project within an organization (business management, information systems management, organizational psychology). Thus psychological factors are complicated by the presence of additional social, technological, and work domain considerations.

Survey of software engineering literature found no academic research comparable to Trist and Bamforth's (1951) comprehensive study, which included discussion of social and psychological issues in relation to a particular production environment and technology. Literature does exist, however, to suggest that a socio-psychological understanding of agile processes could be of great benefit in understanding and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

improving software development processes (e.g. Sawyer, 2004). Warr and O'Neill (2005), for example, discuss creativity and innovation in software development teams through a social psychological lens. In their study they find that while experimental evidence suggests that "real" or collaborative groups are less creative than "nominal" groups that work individually, the dampening effect in real groups can be explained by the social psychological phenomena of production blocking, evaluation apprehension, and social loafing. With this understanding in hand, we can see that agile processes use scheduling mechanisms and group practices such as collective code ownership to circumvent such social dampers on creative and innovative software development.

Teamworking and Self-managed Work Groups

There exists a strong relationship between agile teams and 'autonomous,' 'self-regulating (Hackman, 1976),' 'self-managed (Bryant, Farhy, & Griffiths, 1994; Katzenbach & Smith, 1994),' or 'high performance (Rayner, 1993)' work teams in organizational psychology and management literature. Such teams involve autonomy and self-regulation of relatively large chunks of work activity by members of a small team rather than allocation of individual tasks by a manager or supervisor. High performance teams are closely linked to innovation and the ability to respond to change. These associations, however, are generally based on anecdotal evidence from specific case study research (Hanlan, 2004). Broader empirical research regarding such teams generally finds varied results and some amount of skepticism regarding their effect, particularly on performance and productivity (e.g. Chaston, 1998). The lack of conclusive empirical evidence regarding high-performance teamwork may be related to the fact that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

successful self-managed teams, like successful agile teams, are particularly difficult to instantiate (Hanlan, 2004).

Mueller et al. provide conceptual guidance in an extensive article on 'teamworking' (2000), which they define as "a group of employees, normally between three and 15 members, who meet with some regularity in order to work interdependently on fulfilling a specific task." Mueller et al. take a socio-technical viewpoint when discussing work practices, and strongly advocate sensitivity to context in the study of teamworking. Through a historical view of teamworking they describe how the modern conception of teamwork was discovered and popularized by Tavistock researchers in their observations of industrial work practices in the 1940's (Trist & Bamforth, 1951; Rice, 1958, 1963).

Mueller et al. point out that self-managed teams were not, in fact, invented. The social organization of work through teams arose naturally and spontaneously as a response to adverse working conditions, with workers intuitively adopting teamwork practices which simultaneously satisfied both psychological and task needs (Mueller et al., 2000 citing Trist & Joyce, 1981; Rice, 1958, p. 81). The authors go on to discuss the appropriation of teamworking by the Quality of Working Life (QWL) and 'Humanization of Work' (HoW) movements during the 1960s and 1970s, followed by the teamwork initiatives introduced in the mid-1980's, such as those adopted by Manufacturing Systems Engineering groups. They suggest that the focus of later initiatives on organizational performance measures resulted in marginalization of the democratic and HoW aspects of teamwork (see also Cooney, 2004).

Self-managed teams in the form of agile software practices seem to have been

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

initiated in response to changing work context by software development practitioners, much the same way they were initiated by manufacturing workers more than half a century ago. Comparable initiatives in the open source community (Crowston, Annabi, Howison, & Masango, 2004; Evans & Wolf, 2005) can further be seen. Thus team work processes were not invented by anyone in the arena of management or 'business processes reengineering,' but instead adopted relatively spontaneously from a grass roots level. This indicates that while there may be significant benefits to teamworking in terms of performance, the fundamental driving force behind such flexible team structures rests firmly in the low level realities of human action and reaction within a specific context. Indeed, Mueller at al. further discuss how discourses into work teams and design have been overly concerned with managerial objectives, both in "overestimating the degree to which the organizational context can be shaped by managers," and in underestimating the value of understanding teamworking phenomena beyond performance measures such as quality and productivity.

Mueller et al. conclude by providing descriptions of five dimensions upon which discussion of teams can be based; namely, technological, economic, social, cultural and organizational governance dimensions. The technological dimension includes discussion of the *production technology*, with particular focus on automation in effecting flexibility and choice in production teams. The authors note a positive association between continuous production and teamworking, as well as the effect of production "flowlines" on the autonomy that teams are able to exercise. They also discuss *task interdependence*, with high levels of task interdependence seeming to be a requirement for maximally effective work teams.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Mueller et al.'s economic dimension includes discussion of *teamworking and reward*, as well as the relationship between *teamworking and performance*, which they note is not straightforward and highly dependent of contextual factors. The social dimension of teamworking involved discussion of *employee competencies: skills, training and development*, with distinction between specialization and "multiskilling" in teams. While the cultural dimension involved discussion of the fact that implementation of work groups often comes with a change in management style, and of the importance of peer influences, where "peer control can be seen as oppressive 'groupthink' as well as a constructive 'teamthink' (Neck & Manz, 1994)." The authors reference 'social identity theory' (Tajfel, Billig, Bundy, & Flament, 1971; Tajfel & Turner, 1979) in talking about the importance of a shared purpose and identification with the team. Mueller et al. also explored the *experience of teamworking: attitudes, commitment and 'private lives'* within this dimension, mentioning different forms of employee commitment. Finally, they discuss the organizational/governance dimension, which is somewhat beyond the scope of this thesis.

Computer Supported Collaborative Work (CSCW)

Another research area closely linked with agile processes is that of computer supported cooperative work, which is concerned with the creation of work team environments to enhance or replace face-to-face communication and collaboration. CSCW studies into informal workplace communication seemed particularly promising in providing guidance for this study. Various studies have shown informal communication to be valuable in the execution of work related tasks, the coordination of group activity, the transmission of workplace culture, and in social functions such as team building

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(Whittaker, Frohlich, & Daly-Jones, 1994; Fish, Kraut, Root, & Rice,1993). Other studies have shown informal communication to be useful in dealing with unstructured tasks and uncertainty (Kraut & Streeter, 1995; Curtis, Krasner & Iscoe, 1988). There appears, however, to be little understanding of the exact nature of informal communication in software development or work teams. The focus on behavioral outcomes, particularly, leaves a gap in our understanding of how informal communication affects individual psychology.

Other CSCW studies have shown the importance of awareness and coordination in work groups (Dourish & Bly, 1992) and software engineering (Kobylinski, Creighton, Dutoit, & Bruegge, 2002); particularly, the benefit of informal social mechanisms that naturally facilitate work practices and awareness (Damian, Chisan, Allen, & Corrie, 2003). Again, however, such studies provide little insight into the subjective or social experience of the individual. Research into transactive memory systems (Wegner, 1986) also seemed relevant. These consist of an awareness of who within a team possesses knowledge and skills and support effective use of this awareness (Akgün, Byrne, Keskin, & Lynn, 2006). The 'memory' discussed in such research primarily involves task or role based information, however, with little discussion of social information.

Related studies into distributed cognition (Hollan, Hutchins, & Kirsh, 2000; Hutchins, 1995) and activity theory (Nardi, 1996; Bertelsen & Bødker, 2003) were also informative, but ultimately unhelpful, since they foreground interaction with artifacts and the mechanics of human cognition as opposed to subjective socio-psychological experience. Various other human-computer interaction methods and approaches such as Contextual Enquiry (Beyer & Holzblatt, 1998), Cognitive Work Analysis (Vicente,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1999), and task modeling (Van der Veer , Lenting, & Bergevoet, 1996), have been applied to the design of complex systems and could plausibly be applied to software engineering processes (Sutcliffe, 2005). These approaches still maintain a focus on behavioral and task based interactions as opposed to subjective psychological experience, however, as do most approaches related to software development tools and processes.

Despite inroads into various state of the art CSCW technologies to enhance the communicative and collaborative experience, work teams are often likely to resort to flexible plain text mediums in the absence of face-to-face interaction (Evans & Wolf, 2005), effectively passing up structured communication mechanisms in favor of unstructured ones. Such activity illuminates the limits in our understanding of the social nature of human interaction as we attempt to create systems that support the intricacies of the social environment at work, as well as the practicalities of work tasks.

Summary of Conceptual Framework

This section framed an agile software development team as: a socio-technical system; a complex adaptive social system; and a self-regulating team. All three aspects of the conceptual framework highlighted the importance of viewing agile teams as holistic entities operating within the context of software development. The combined frameworks imply exploration of agile teams as consisting of individuals, the group, agile practices, and the socio-psychological characteristics and relationships between them.

The socio-technical perspective highlighted the importance of considering agile teams as systems comprised of both social and technological components. The use of physical artifacts in agile, for example, such as interactive wall charts and automated testing tools, seem integral to team coordination and motivation. It should be noted that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

agile technology as defined in this study, encompasses all agile practices. Techniques such as the use of information radiators to communicate information, and interaction with working software through test-driven development are obvious technical components of an agile socio-technical system. Perhaps less obvious as technological components are practices such as daily stand-up meetings. These practices structure and mediate human action and interaction just as physical technologies do, and were seen in this study as comparable.

The systems perspective highlighted the value of viewing individuals as subsystems interacting within the larger team system; thus agile practices can be seen to not only help individuals work within their limited capacities of memory, perception, and action, but to allow teams to work together as interconnected and cohesive units. The importance of social processes, and the differentiation of a group entity, were suggested by exploration of human agency in systems theory, as well as by research underlining the importance of awareness (Dourish & Bly, 1992; Kobylinski et al., 2002), social awareness (Erickson & Kellogg, 2000), transactive memory systems (Akgün et al., 2006), shared work rooms (Teasley, 1996), and the effect of the collective on individual psychology (Bion, 1961).

The systems perspective also suggests the value of observing feedback and feedforward cycles in agile teams. This can be associated with the checks and balances that agile practices place on the process of software development, placing attention on processes that increase 'good' software development practice, and decrease 'bad' software development practice. For example, in light of business needs in the new economy, good practice is responding to change.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finally, viewing agile teams as self-regulating teams highlighted the importance of studying subjective, and non-task-based experience. Just like self-regulating production teams fifty years earlier, self-regulating agile teams resulted from a grass roots push by practitioners. Agile practitioners went a step further, by taking techniques that had served them well in the trenches and welding them into a cohesive and business savvy software development methodology. The origins of the agile movement suggest, however, that traditional management concerns, such as task-related behavior and performance, are inadequate for describing interaction within such a methodology.

The preceding sections outlining the research domain and conceptual framework have highlighted issues that seemed particularly relevant to a study of experiences in agile software development teams. The following guiding questions, therefore, were referred to during the course of this research:

- What are the basic psychological needs that are satisfied by agile software development?
- How do agile practices affect individuals in terms of positive psychology – e.g. in increasing meaning, self worth, motivation, excitement, and flow?
- How do agile practices increase social awareness?
- What social forces are at work in team-based software development?
- How do agile practices mediate interaction between individuals and the group?
- How do agile practices help align individual (subsystem) and team (system) goals?
- What kind of information is being passed between individuals in an agile team and how? What information is being amplified and how? What kind of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

information is being muted and how?

- What emergent properties exist in an agile team environment?
- How do agile practices help teams manage interdependencies?
- How do agile practices help with team coordination and scheduling?
- How do agile practices help individuals and teams self-regulate?
- What informal communication mechanisms are being used? What formal mechanisms?
- How does agile help individuals deal with the complexity and uncertainty of team-based software development?
- How do agile processes help individuals adapt to change?

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Research Method: Grounded Theory

What is Grounded Theory?

Grounded theory (Charmaz, 1995; Glaser, 1992; Glaser & Holton, 2004; Glaser & Strauss, 1967; Strauss & Corbin, 1990) is a methodology that provides a set of procedures for the systematic collection and analysis of qualitative data. It is a process by which a theoretical account of a topic is systematically abstracted and developed through examination of, and grounding in, empirical data.

Grounded theory research focuses on exploration and theory generation rather than theory verification (Glaser & Strauss, 1967; Rennie, Phillips, & Quartaro, 1988). It is characterized by use of the constant comparative method of analysis, with data and abstract concepts constantly compared to each other, ensuring the development of an integrative theory that is firmly grounded in raw data. Data collection and analysis are conducted simultaneously, with emerging analysis and theory shaping subsequent data collection.

Why Grounded Theory?

The complex process of software engineering seems best suited to naturalistic qualitative research. Traditional research methodologies have not been sufficient in understanding the complex interactions occurring in today's business and software development environments (Fitzgerald, 2000; Nandhakumar & Avison, 1999; Russo & Stolterman, 2000). Quantitative or experimental methods derived from the natural sciences have also been found to be "inadequate and inappropriate" (Lee, Liebenau, & DeGross, 1997) in attempts to explain the human and group issues related to technology

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

use, particularly in that they focus on cause and effect models of interaction (Merali, 2004). Increasingly, researchers have begun to address this issue by exploring practice in a naturalistic setting through qualitative data (Baskerville & Stage, 1996; Myers & Young, 1997; Nandhakumar & Avison, 1999; Orlikowski, 1993; Wastell, 1999). Grounded theory allows for the study of phenomena, such as motivation and excitement in software development team, that are not easily observed in classical research. As Rennie et al. (1988) note, "exception from the constraint of the verificational approach frees grounded researchers to address highly complex meaning structures that might otherwise be beyond the pale."

Curtis (1984) further proposes that the main problems in conducting psychological research of software engineering stem from the difficulty of integrating applicable theory and data gleaned from a necessary assortment of paradigms in psychology. The study of software engineering processes is a multidisciplinary endeavor, with both socio-technical and team perspectives involving rich interactions and emergent effects that are ill contained in traditional disciplinary paradigms. Grounded theory effectively crosses disciplinary boundaries (Glaser & Strauss, 1967), bringing together relevant aspects from different fields of literature in the construction of a new theoretical framework. Exploration of interactions between individual and group levels of analysis, for example, can be encompassed in grounded theory research. Grounded theory thus offers an alternative to awkwardly transposing psychological research directly onto software engineering practice, allowing isolation and abstraction of constructs that can then be related back to existing theory, or provide avenues for future study.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finally, the results of grounded theory analysis remain firmly entrenched in data gathered from the field of practice. This is particularly valuable in disciplines such as software engineering methodology, information systems, and human-computer interaction, which exist very close to their practical applications. Zelkowitz, Yeh, Hamlet, Gannon, & Basili (1983) note how field studies examining software engineering tools and methods can detect discrepancies between state of the art and actual practice. This study utilized grounded theory to discover such discrepancies in terms of the social and psychological processes affecting human experience in agile software development teams – topics that are not fully discussed in agile literature to date.

A cross-cutting research area, scarcity of socio-psychological studies regarding software development and engineering, and lack of guidance as to which theories from psychological literature were relevant to this study, led to the proposal of a qualitative grounded theory study of agile phenomena. While the conceptual framework of this study provided a clear research focus, grounded theory methods allowed for diversion and exploration around these themes. Grounded theory was therefore seen to offer a structured method by which to discover aspects of agile methodologies that were important and relevant to agile practitioners, as opposed to prematurely focusing on preconceived notions.

Research Perspective

The original grounded theory researchers (Glaser & Strauss, 1967) diverge somewhat in their ideas of grounded theory as a process of theory discovery (Glaser, 1992) or theory construction (Strauss & Corbin, 1990). This research subscribed to the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

interpretive view of Charmaz (1990, 1995), which sits somewhere between these two extremes.

While theory is perceived to emerge from the data, the perspective and understanding of the researcher plays an important role in the kind of theory that unfolds (Charmaz, 1995). The researcher's world view, disciplinary training, theoretical leanings, and research interests have an effect on observations and treatment of the data, and the types of questions asked by a researcher can generate data related to specific aspects of experience that participants may not have explored without prompting. Rennie et al. (1988) additionally note how each researcher controls the scope of their research.

The theory elicited by this study, therefore, was bounded by the view of agile teams as holistic systems, within which the subjective experience of the individual was examined. The scope of the research was defined by a focus on the relationships between system components, with initial attention paid to aspects of agile methodologies supporting cohesive teamwork and individual motivation and excitement within teams. That said, efforts were made to avoid pushing preconceived ideas or theories onto data (as discussed in Glaser & Strauss, 1967), and to remain open to other socio-psychological incidents more closely related to the realities of agile software development. The aim of this grounded theory study was to discover the main concerns of agile team members within the scope of the research questions, and how these concerns are resolved (as discussed in Glaser, 2004).

About the Researcher

Interpretive analysis is highly dependent on researcher values and beliefs, which are generally understood to be shaped by life experiences (Creswell, 1997). This section

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

will therefore outline professional and personal experiences that seem likely to affect the theoretical sensitivity (Strauss & Corbin, 1990) of the researcher in the development of a grounded theory.

I have an undergraduate degree in human-computer interaction from New Jersey Institute of Technology. The degree included the equivalent of a degree in psychology, and classes such as user-centered design, information systems, management, computer-mediated communication, systems analysis and design, and the basics of programming. I have worked for a year as an intern in usability and user centered design. During this time I was involved in the design and development of internal web-based applications within a large organization that followed the Rational Unified Process of software development. I am a proponent of agile software development, and have actively participated in the agile software development community for just over a year.

Conducting the Study

Study Participants

This study involved 22 participants recruited through networking with members of the agile software development community. Participation was voluntary, and secured on an individual basis. The information provided to participants is outlined in appendices B and D.

Attempts were made to gather different perspectives of agile methodologies as represented by different team roles, with particular attention paid to the role of usability or user centered design specialists. Participant roles included five agile developers, three interaction designers, one user experience manager, five project managers, two agile coaches, two quality assurance specialists, one technical documentation specialist, and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

three developers who had previously been project managers. Four of the interviews (two quality assurance specialists, one interaction designer, and one developer) were not coded, 'memoed', or explicitly included in study results. Data from these interviews did, however, inform theory development. All but two of the participants had previously worked in non-agile teams, and five of the participants were transitioning from an agile into a non-agile development environment at the time of the interviews.

There were sixteen male participants, and six female participants. Participants came from a range of organizational and team structures; from large multinational companies to small start-ups, and from entirely self-regulating teams to teams with high levels of management supervision. Five of the participants in the study were from the same software development team. Organizational environments, along with participant demographics such as age, education, and gender, were considered, but were not maintained as a focus of this study.

Data Collection

Data was collected through semi-structured one-on-one interviews with members of agile software development teams, through interaction with the agile software development community, and through survey of relevant literature and artifacts.

Semi-structured one-on one interviews

Participant interviews sought to investigate the subjective experience of individuals in agile software development teams ('agile practitioners'). Semi-structured interviews were chosen as opposed to structured or non-structured interviews in order to maintain focus on the theoretical framework, while still leaving room for phenomena significant to the interviewee to emerge. Interviews, which were conducted over a period

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of two months, ranged from half an hour to an hour and a half long, and each interview was audio recorded and transcribed. The initial semi-structured interview guide (Appendix C) evolved as the study progressed based on coding, memoing, theoretical category development surrounding the questions highlighted by the theoretical framework, and general impressions between interviews. Thus theoretical sampling (Glaser, 2004), or theory-based data selection (Rennie et al., 1988), was utilized to reflect fruitful areas of study emerging from data, or gaps in understanding highlighted by theoretical analysis (see Strauss & Corbin, 1990; Charmaz, 1995).

The progression of each interview was highly dependent on the nature of the experiences and responses of each participant, and discussion in interviews involved a wide range of experiences surrounding software development team activity. Interviews would generally involve one or two teams as a focus of discussion, with experiences from other software development teams referred to when relevant. Personal experiences and/or team activity were discussed, particularly those related to satisfaction, motivation, and excitement. Project characteristics such as domain, team composition, physical space, and agile practices employed, were elicited whenever possible. Initial interviews focused on participant comparison of cohesive and non-cohesive team environments and the differences between them, as well as comparisons between agile and non-agile team environments. Later interviews focused on experiences surrounding division of responsibility and aspects of agile software development that participants found to be related to personal satisfaction and team cohesion.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Other Data

Survey of tools and artifacts. A number of tools are used during agile software development, including tools to automate and support development, collaborative software development environments, and various mechanisms for project management and communication. Attempts were made to gain a basic understanding of the environments within which software developers were working, and to understand the relative efficiencies and deficiencies associated with the tools used during development. Artifacts related to my area of study included project story-boards and information radiators, automated testing frameworks and tools, prototyping and design tools, and software products under development. Understanding of such tools was gathered in discussion during interviews, through informal observations made during visits to agile work places, and through reference to relevant literature.

Survey of literature. Literature in grounded theory was not given priority over other data, and was accessed as it became relevant. Two types of literature were considered applicable during this study. Firstly, experience reports and practitioner literature from the agile community were accessed to validate findings and add to the data set. Since practitioner literature is written for an audience, however, and from a management standpoint, it was considered secondary to interview data. Secondly, academic literature as outlined in the literature review was referenced through the course of this study.

Participation in the Software Development Community. Insight and data were also gathered from study of, and participation in, the agile software development community. Such involvement included participation in agile conferences and workshops,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

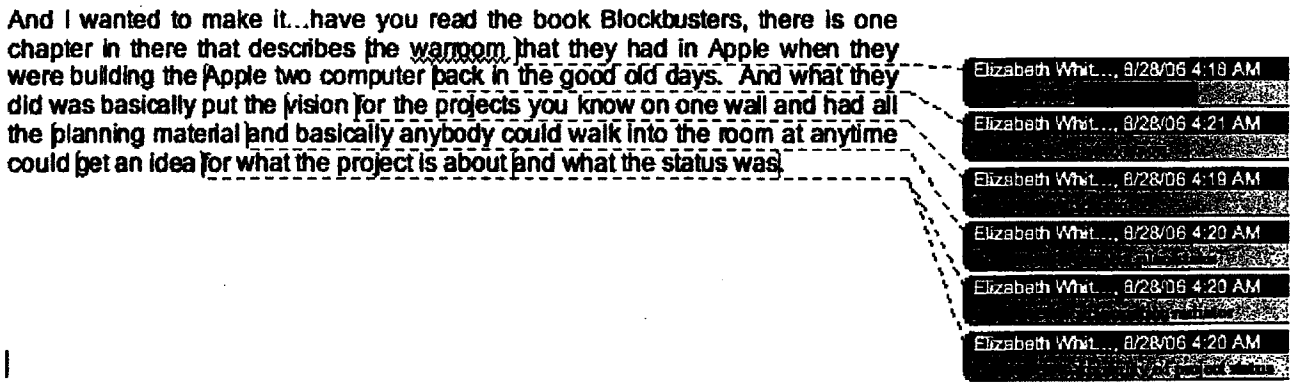
involvement in agile user groups and mailing lists, and general discussions with software development practitioners.

Data Analysis

Open and Axial Coding.

After initial data collection, data was broken down into discrete parts and incidents were identified, conceptualized, and named in a process called open coding (Strauss & Corbin, 1990). Open coding was conducted line by line to ensure thorough grounding and critical thinking about the data (Glaser, 1978). Early codes were highly dependent on interview data, while later coding was somewhat more focused on emerging theoretical categories. Initial codes were considered provisional (see Strauss & Corbin, 1990), and provided guidance as to where the study could be taken using theoretical sampling (see Glaser, 2004; Charmaz, 1995).

Example of coding:



Axial coding (Strauss & Corbin, 1990) was then used to examine the relationships between data. Strauss and Corbin suggest a coding paradigm in which conditions, context, action/interactional strategies, and consequences are explored and defined for each category. While Glaser (1992) holds that this unduly forces theory into

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

preconceived categories, there seemed to be value in the paradigm, and it was used to the extent that it fit the data gathered. Codes that occurred repeatedly or seemed important in the research context were written on post-it notes and placed on a wall in categories that shifted and changed as the study progressed. Properties and dimensions for each category were constructed as outlined by Strauss and Corbin (1990). Open and axial coding were performed in parallel as data were gathered, analyzed, and reanalyzed in light of the emerging theory or concepts.

Memoing

Theoretical concepts and ideas that occur during the grounded theory process were systematically recorded as memos. Thus a record was made of hunches and hypothesis to be proved or disproved by theoretical sampling and the constant comparative method (see Glaser & Strauss, 1967; Strauss & Corbin, 1990). Each memo was dated, referenced back to the raw data which sparked the insight, and included a heading denoting the category/s or concept/s to which it pertained (when applicable).

Initial memoing generally occurred during the coding process, with memos written in relation to the interview they were referring to.

Example of Memoing:

Interview: T#1, July 28th; Coded: August 5th

T.1.5: [redacted] **again, the war-room concept brings to mind visions of a hive of activity – a place where *stuff happens.*** Decisions are being made, things are being built, people are doing stuff. **What effect does being in this hive of activity have on the people in it?** The visibility and awareness of *stuff happening* seems to be important.

Relating this back to the buzz that people feel when in this kind of place. The buzz usually seems to be really present when people are working towards some kind of goal...**Why does this buzz seem to depend on the presence of a project goal? Is it because of the fact that they are working towards a goal?**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Or is it because people are just more active when there is an upcoming deadline and therefore the environment is more exciting?

Memos included color-coding as follows:

[redacted]: interesting observations and directions for future thought

Yellow: directions for further inquiry/question

[redacted]: core research issue

[redacted] woot (very important)

Written Analysis of Data

Later memoing involved the construction of the Analysis of Data document (see Appendix F) where each interview and related initial memos were systematically integrated into the document. This involved taking memos and coded interviews for each interview and selecting codes, quotes, and memos to be included under categories of participant experience. A version of the document was saved after the inclusion of each interview to show the progression of theory development. In addition to the comparisons that occurred during initial memoing, closer inspection and comparison of codes across interviews and situations occurred at this point, as well as checking between theoretical categories and initial situations from which codes were generated. Interview quotes are referred to by a reference number throughout the document, for example, (L.4.35). The three radixes refer to the interview batch, the number of the study participant within each batch, and the interview paragraph from which the quote was taken.

Theoretical Saturation

Charmaz (1995) notes that most grounded theory researchers today do not emerge from their studies with a substantive or formal theory, but instead come away with a conceptually rich analysis of the environment and lived experience of participants. Thus

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this study put an emphasis on developing valuable analytical categories to explain phenomena as opposed to a tightly woven theory.

In a full grounded theory study, a core category would be selected and theoretical saturation sought under this category, where sub-categories are seen to be theoretically dense, and the relationships between categories are well established and confirmed (Strauss & Corbin, 1990). A short time frame for this study, coupled with the broad research area, meant that theoretical saturation was reached only in very general terms. Results include a high level understanding of the ways in which agile methodologies structure and mediate team activity and result in positive individual experience. While relationships observed in the study emerged and were confirmed with the data, detailed understanding of the properties and dimensions of categories across situations requires future study.

Confirmation of Results

Glaser (1992) offers two major criteria for an emerging theory: that it fits the situation, and that it helps people in the situation make sense of, and manage their experience better. The first criteria was ensured by "purposeful grounding" in data (Strauss & Corbin, 1990) and the method of constant comparison (Glaser & Strauss, 1967; Glaser, 2004). Abstract concepts were tested against data, and only validated concepts appeared in final study results. Good fit was also ensured by frequent discussion and elicitation of feedback regarding the evolving theory with members of the software development and agile communities. This also ensured adherence to Glaser's second criteria, namely, that the research report be accessible and useful to agile software developers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The written Analysis of Data document and Study Results were sent out to participants to elicit feedback on the validity and completeness of data and interpretations, a number of whom replied to confirm the findings. Documents were additionally shared with several other members of the agile software development community, who were satisfied that results represented their experiences in agile teams. Feedback suggested that results resonated with readers, presenting a persuasive theory that provided both relevance and guidance, in accordance with requirements for credibility in grounded theory studies (Glaser & Strauss, 1967).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Study Results

Introduction to Results

The motivation for this research was to better understand the animation and excitement observed in practitioners of agile software development. The initial research question for this study asked: *what is the experience of being in an agile software development team?* The question was further structured to consider: How do agile practices structure and mediate the experience of individuals developing software? In particular, how do agile practices mediate the interaction between individuals and the team as a whole? And what are the socio-psychological characteristics surrounding agile practices and individuals within a software development team? In particular, how do agile practices contribute to positive feelings such as motivation and excitement? It should be remembered that the view of agile in this study is based on, but not limited to, extreme programming practices (Beck, 2000, 2004).

Participants in this study, when asked about software development teams characterized by strong feelings of excitement, discussed well functioning teams that 'clicked,' 'gelled,' or '*really* worked together' to successfully develop software. Such 'cohesive' teams can be held in comparison to non-cohesive teams, which were not associated with feelings of excitement. One of the main sources of individual motivation, excitement, and satisfaction stemming from agile software methodologies, was found to be an improved ability to truly work together in such teams to develop small software increments; particularly in relation to traditional methodologies, which were more likely to prescribe in advance how team members were supposed to work together on a long term basis. Study participants in agile environments showed related increases in their

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sense of security, control, pride, and ownership over both the software product and the team processes used to create it.

The enhanced ability of team members to work together to develop software as a cohesive team was seen as dependent on team-wide buy-in to shared and holistic goals, the importance and presence of these goals in day-to-day team action and interaction, and a shared understanding of the actions that needed to be taken in order to reach these goals. The distinction between cohesive and non-cohesive teams in this study was separate from the distinction between agile and non-agile teams. Examination of results therefore included the question: what characteristics of agile teams are related to team cohesion?

Agile Environment

A 'typical' agile team consists of a small group (4 to 12 people) working in a *shared team room* with few, if any, barriers between them. Large desks are set up to allow two or more team members to gather around the same workspace or computer screen. The physical environments described by participants in this study were extremely diverse, however, with well functioning and cohesive agile teams existing that were dispersed across an organization or city. Geographically dispersed teams were discussed as points of comparison to teams that could meet face-to–face on a regular basis, but were not a major focus of this research.

The agile practices considered important to team functioning by participants were also diverse. The majority of teams, however, were seen to function around the *agile planning game*. This involved identifying business value in the form of features, or 'stories', developers estimating the time needed to develop each story, and the team

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

choosing stories to be included in each iteration, or cycle, of software development based on prioritized business value. *Information radiators,* or highly visible and shared team artifacts placed on a wall or whiteboard, were used ubiquitously by agile teams in this study, although to varying degrees. Project activity was usually centered around a 'storyboard' or 'iteration plan' information radiator, which would clearly display to the team the stories to be developed for the current iteration of development. While the exact nature of such information radiators was often unclear, the level of detail of their content was quite high. Iteration plans would often indicate day-to-day operations of agile teams, including which team members were working on each story. The iteration plan was sometimes part of a larger information radiator, which would indicate upcoming and completed stories beyond the scope of the current iteration, or team and project issues outside the scope of development.

Most teams practiced *continuous integration*, where development output was integrated into a working software product on a regular basis. The majority of teams were also seen to practice *iterative delivery*, where a comprehensive version of the software product was produced in short iterations of development, such as every week or month. The agile practice of *pairing*, where two team members work together on the same task, was adopted to varying degrees by cohesive teams on this study, but was shown to be highly significant in increasing team cohesion. The most consistent practice discussed surrounding cohesive teams was the *daily stand-up meeting*, where all team members would gather for 10-15 minutes every morning. During such meetings each team member would report his or her progress on their current task, any problems or issues they were having, and any future needs related to the rest of the team.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Making Teamwork Easy

> *And so once we started adopting some of the agile techniques in my previous product I was working on, they were very welcomed. It was instantly recognizable as a pleasant way to work for the people, for the developers, for the managers, for everybody involved; because there is less crisis, it's easier to manage, you have a better idea of what it really takes to deliver what you are - what people are asking for. It's so much more manageable...I mean it takes out uncertainty, it reduces the risk, crisis management, it's easy to schedule and plan – everyone's happier.*
>
> *(T.3.33/44)*

One of the main factors associated with enjoyment and excitement in agile software development teams in this study was the ease and speed with which team members could get things done; questions were answered, problems were resolved, and collaborative opportunities were quickly grasped. These interactions were seen to be reflective of a number of factors supported by agile software methodologies, including: instantiation of agile culture and values; team-wide commitment to a collective goal; the agile planning game; regular iterative delivery; process improvement; continuous integration and automated testing; frequent and informal interaction in the team environment; whole team involvement and awareness; and the relative freedom of team members to interact or self regulate within the scope of the collective plan. These points will be discussed at more length throughout the rest of this document.

Agile Culture

Activity in the software development environment was seen to be highly mediated by team expectations; particularly by way of a tangible 'agile' culture, adopted to a greater or lesser extent across ostensibly agile and non-agile teams. Interview participants who strongly identified themselves with the agile culture of software development ('agile participants') exhibited common understandings and values. These existed relatively

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

independently of the particulars of software development methodology or practices, although agile practices were seen to be highly supportive of such culture. Interviews revealed the presence of an 'agile ideal' regarding both individual and team activity that most agile participants were striving to live up to. Though exact interpretations of 'agile' are many and varied, the core of this ideal for most participants seemed based on editions of *Extreme Programming: Embrace Change*, by Kent Beck (2000, 2004).

Three characteristics associated with agile culture seem particularly pertinent to this study. Firstly, agile teams were seen to encourage an environment of openness, honesty, trust, and respect. In particular, participants in agile teams showed a heightened awareness of the actions and opinions of others in their team, and a better understanding of their own opinions and roles in relation to the rest of the team. This reflects an essential aspect of cohesive team environments observed in this study, which was *transparency*. While agile practices instantiate team activity to be observable or transparent, cohesive agile teams were seen to ubiquitously encourage openness and honesty, from developers to management. It is expected in such teams that problems, issues, points of contention, and points of interest, will be brought up immediately and discussed openly. As one agile team leader noted:

> *This isn't a place that you go and hide*
>
> (O.2.14)

Many agile team members were seen to express feelings of liberation, and no small amount of relief at being able to develop software in such an environment.

Secondly, agile participants showed a strong inclination for whole team consideration and involvement. Participants in agile teams would overwhelmingly talk

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

about 'we' and 'us' rather than 'I' or 'me,' and would talk in terms of holistic and systemic visions of the software product or process rather than in terms of individual tasks or roles. Agile participants were seen to be particularly concerned with involvement and inclusiveness across the entire software development team. Such concerns were supported by agile practices, which allow for ease and speed of communication and collaboration, frequent interaction between team members, and high levels of immersion in project activity:

> *Throughout the project, whenever you think that you need to find anything out at all you just go and talk to people. That's another thing that is very different in agile, is that all roles are involved throughout the entire process.*
>
> (L.3.20)

Finally, agile participants placed high value in action, initiative, and continuous improvement. Participants involved in cohesive agile teams, particularly, discussed change in a very immediate sense. They talked in a way that was much more active, probing, and inquisitive, and tended to display a kind of exploratory idealism regarding the changes that they could make to improve their team and project environments. Agile participants in general also showed a tendency to talk in terms of actions to be taken in the near future that would complement ideas or thoughts that came out during the course of a discussion or interview. In comparison, non-agile participants showed a lack of active participation in their working environment, and even a certain degree of submissiveness and lethargy regarding the change that was possible in their team or organization.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A Clear Objective

> ***Have you been on a really dysfunctional team?***
>
> *In some ways that spec driven team was a bit dysfunctional, but that might have just been my perceptions. Um – its bits where there's more push to meet the spec than to meet the customer needs. I mean, in my view that's dysfunctional, but in terms of most software engineering teams, it's not.*
>
> (L.1.82)

What was found to be one of the most valuable aspects of agile software development methodologies in supporting cohesive teamwork was that they provide a clear team goal – to deliver the most business value to the customer in the shortest amount of time. Cohesive agile teams also maintain a strong focus on developing *quality* software code, above and beyond the needs of the customer. The value of such goals from a team perspective is that they are goals that most everyone can agree on.

The Planning Game

In addition this team goal, the most value in the least time, is instantiated in a clearly visible and rigorous process as outlined by the agile planning game. Participants often noted the agile prioritization and planning procedure as a point of pride in their team process, and expressed feelings of excitement in that it allowed them to negotiate to create a *plausible* plan to develop software:

> *And then the other thing where there is a lot of tension is just when you develop a feature there is only so much you can do, you can't - well the product specialist will often come in with a list of a 1000 items on it. And the developer has - there is this dance that takes place and the developer estimates, they say well I can do 50 of those. And you know that basically boils down to the acceptance criteria that get used and you know, frequently the acceptance criteria will include things that are just impossible to do. And so we will have discussions about that a lot earlier and that also causes a little bit of tension because you know, now the product specialists realize the implications of all they are asking for. And then likewise the developers realize that they actually do need to put*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *some thought into how they are going to do something so that they can provide reasonable feedback and reasonable estimates and things like that.*
>
> (T.1.11)

In particular, the agile planning game makes very clear the constraints of development, and was seen to give developers comprehensible bargaining points against attempts to squeeze additional features into a software product. The construction of a reasonable plan was related to a reduction in stress and increased sense of control over the development process when compared to traditional planning methods, which were discussed as often involving unrealistic expectations in terms of development.

The use of story cards placed on a board with a limited amount of space was seen as especially effective in bringing home awareness of team constraints, and in making inherent role conflicts easier to digest:

> *We've only got five weeks but we've got ten weeks worth of work to do here so something's got to give. We take five of that and we push it to the next cycle or, and re-adjust that theme or, you know, everything has an effect and you can see that effect on the big board when you do the planning.*
>
> (T.3.18)

Thus the planning game and associated information radiators were seen to increase awareness of the implications of business decisions on the task of software development. Participants noted that this was especially valuable as a means of generating agreement, and the process was seen to greatly reduce the tension and conflict traditionally surrounding requirements specification and planning.

The agile environment was also seen to promote team participation in plan construction, thus supporting individual involvement, engagement, and buy-in to project planning and activity:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *And the product owner who takes the items that have been called out as priority for the customer and says 'I think this one's next, this one's next, this one's next.' And then the team looks at it collectively and says 'Okay based on our knowledge this is how long it's going to take.' And so there is negotiation back and forth, they say 'Well I want these three' and then the team says 'Well with this sized team, it's going to take us 60 days, we can't do it in that amount of time,' or 'We need to break them out.' So it becomes this negotiation between the representative of the customer and the representative of the product that's going to be built. So it's a very open collaborative effort to decide.*
>
> (I.1.14)

> *It's about buy-in. So if I get handed a task and they say 'can you do this task please and you need to have it done by the end of the week' then I will try and do that. But if I haven't estimated that myself and there is a tight deadline I'm not going to take the responsibility to go and actually meet that deadline, because it wasn't me that came up with those sorts of timeframes. But if I came up with a timeframe myself, and estimated it myself, then I kind of feel obliged to do what I said I'd do. And also, you get to choose what interests you. You get to work on what you want to work on, Rather than what they tell you to work on. It's all about buying in to the overall goal, rather than this is my task, this is your task.*
>
> (L.3.4)

The ability of the agile plan to adjust and allow for changing business needs and also for individual capability and current team activity, was seen as essential to team certainty and agreement surrounding team activity. Customization of the plan to specific project and team needs was particularly related to cohesive team activity. The agile practices of allowing developers to estimate their own stories, for example, and the fact that the project plan would then be constructed around these estimates, seemed associated to the feeling of 'rhythm' or 'flow' often discussed with regard to agile teams. Planning in this manner was seen to allow individual pace and team pace to be highly in synch, where tasks to be completed were neither too difficult nor too easy for individuals in the time allotted. Such team momentum and responsiveness was highly related to excitement and motivation in the agile team environment. In contrast, teams that adhered to a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

specified plan despite changing project requirements were seen as unable to support flow or momentum from the standpoint of the individual.

Flexibility in the agile planning process was seen to have additional effects, particularly in contrast to long term planning. Long term planning was associated in this study with high levels of individual investment in planning artifacts, inclusion of non-essential software features, and forced or premature commitment to design decisions. Participants discussed a tendency, when working in long term planning situations, towards inclusion by default rather than exclusion by design:

> *What I have done in the past has been different, like at other companies where it was more waterfall where you created a obnoxious UI stack from now until next year; and then stuff changed, and I am not a big fan of that. You know there is a lot of investment in terms of writing stuff out and then once you have done all that, the commitment is really high to keep all of that in, even though it may not be the best solution.*
>
> (T.5.5)

Long term planning was also associated with pressure for individuals to ensure that their specific needs were met in a plan that they would have to live by for the course development. In comparison iteration based plans spanning a week, two weeks, or a month, seemed to increase the perception of the current situation as temporary or non-fatal. Flexible and short term planning was therefore seen to allow for more relaxed team relations in planning and implementation, seemingly because there is less at stake. An interesting finding, for example, was that participants were more willing to put up with less than ideal situations or less enjoyable work if they knew that it would only be temporary:

> *Yeah there are a lot of people in the team that really don't like some of their team members. But let's face it, they can pair with them for a week.*
>
> (O.2.21)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reduced preoccupation with personal tasks or goals not directly contributing to the goal of developing working software was also seen to allow a team focus on generating agreement and succeeding in team-based software development around a small number of tasks. Such focus was additionally reinforced by social accountability and recognition in the agile environment, as will be discussed later. The ability and expectation for change in the agile environment, in conjunction with the construction of a plausible plan for development, was thus seen to reduce worry, stress, misunderstandings, and strained team relations resulting from long-term planning processes.

The proposed plan in agile teams is regularly realigned to reflect the true state of the project and what can credibly be achieved:

> *The way that we are working now there are very few if any crisis. That was not the case with other projects I have been part of where we didn't use this type of process. There could easily be a crisis – that [you] couldn't get the features in; you ran out of time; there was something really buggy...With [agile] because the cycles are so short and there is so much revisiting of the plan it really removes a lot of the ability for a crisis to appear because you know right away that something is going to take a whole lot longer or something is not working...and you can reschedule and you can refit everything accordingly.*
> (T.3.32)

Crisis, in this participant quote, reflects an inability of the team to work together on a specified plan, rather than a lack of ability to develop software features. The plan specified in cohesive agile teams was seen to reflect team activity and progress more accurately than in non-agile teams, and to instantiate a culture of interaction for current team activity that was more likely to be agreed upon, or at least accepted, by all parties. Such buy-in by all team members to the shared team plan was seen to further support individual commitment to the project, and thus support higher levels of involvement and excitement regarding project activity.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Lack of flexibility in team planning, on the other hand, was related to low levels of agreement and certainty in the team environment, and some amount of dissatisfaction with project activity. Disparity between the proposed plan, the 'official' project status, the actual activity in the project, and the 'best' solution, was related to increased divergence in team member vision and purpose; to the point where team communication was no longer about how to reach a team objective, but about what exactly was going on and what they were trying to achieve. The confusion resulting from such divergence was further related to an inability for team members to become engaged with the problems at hand:

> *You know, often times as a developer on a traditional project, there is just so much chaos that you don't get to create. And you consequently write poor quality software, because you are constantly thinking about – 'Oh my gosh we are not going to be done on time!' 'This project is a disaster!' 'I don't know what this requirement is!' instead of kind of going – 'What if we did this...?'*
>
> (X.1.12)

Interestingly, while the agile plan was seen to highly regulate individual behavior over the course of an iteration, the relatively constrained nature of the agile environment was related to feelings of liberation:

> *It's invigorating, because for the first time in your career you know exactly what is expected of you.*
>
> (O.2.15)

Regular Iterative Delivery

This study included some interesting observations surrounding the effects of iterative or priority based development in the agile environment. Firstly, iterative delivery was seen to increase the sense of immediacy in the team environment; particularly through the prioritization process, which meant that the majority of the team would

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

always be working on the *most important thing*. The fact that team members had to deliver a working product at the end of the week or month was further seen to increase the sense of urgency in team interactions, and the capacity for team members to resolve or put aside personal differences in order to work together to *deliver*.

The ability to deliver regularly in agile software development was placed in particularly high regard by developers, and was several times noted as the main motivator related to agile software development:

> *You are working on something and at the end of the week it goes out to the customer and you get feedback right away. And that's great, because your work matters; every day matters. You notice when we have a new product and you are working on it for 6 months and then it is really tough going; because you are not delivering.*
>
> (O.2.8)

> *Traditional teams just aren't as focused. In an agile team, because you are working such tight, such short iterations, you are showing something for your efforts fairly regularly, and that's quite a buzz.*
>
> ***Can you describe in more detail this buzz?***
>
> *There is kind of a, a slightly lifted intensity. When you're delivering something every week or few weeks. Whereas in a traditional model you might be going for three months without delivering anything; and when you do deliver something it doesn't often work and there is nothing to show for it because they tend to do horizontal slices. You will develop a component that is useless on it's own, so you have nothing to show for it once you develop it. Agile you are always, you are developing software that people can take it and play with it and do something. Perhaps it is not everything that it should be doing, but it's still doing something that they can see.*
>
> (L.3.1-3)

Agile practitioners, in delivering working software on a regular basis, were increasingly able to see the *purpose and value* of their efforts outside of the context of development tasks, as well as outside of the context of project activity. Holistic understanding in the larger context of development was seen to provide meaning to low-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

level tasks, and to support the ability and willingness of team members to create software that would be of value to customers and users:

> *You know I am trying to think back in the old way when I was on [another larger project]. I think those user issues were very much filtered. So by the time it got to me, it was very much more explicit how it was going, what it was going to do and how it was going to behave; and not a lot of background as to what the original task was and what the original situation was, [which] makes it harder to question...Whereas I think definitely on this product there is a lot more interaction that way and we can go back and forth and say 'You know I know a customer said he wanted this but I think really it would work better if we gave him this,' which to me makes a better product definitely. I really got the sense of that being a problem in [on the old team]; where you work, you work and work and work on a feature, in the end it isn't exactly what they want but at that point it's too late; and I think in agile you at least modify it on the way and get, end up with a better solution. Does that motivate me better? Yeah I think so.*
>
> (T.2.48)

> *As far as the developers went, I don't know, I found the developers much more interested in the project and much more interested in what we were doing because the tight feedback loop, they could actually see the difference. Like they could, I think most developers want their stuff to be successful. It's just that they are normally not given the information to make it successful. Like they don't want to make a bad UI, right - they want to make a good UI, they are just not given the information. So when we give them the information they could change something and then we could come back and say, 'The users are – they want the prototype, they don't even want to wait. They kept trying to pull it from us right.' That makes them happy too. So we had a nice relationship there that we could give them the feedback that they were really – that they really wanted.*
>
> (T.4.28)

Thus splitting the planning and development activity into small chunks was related to increased motivation and enjoyment surrounding project activity as a whole:

> *On other projects or teams that I was on, it was very much you define a bunch of features and then you worked on them for months and then you bug fix for twice as long and that was the release. So you really didn't know what the big picture was, ever. You just knew that one of these or a couple of these features were what you were supposed to work on for however long and then you are just fixing bugs for ten months or – well that's an exaggeration, but that's pretty much what it felt like.*
>
> *(T.3.18)*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Such chunking of activity was also seen to allow for focused discussion and implementation of progressive steps of software development, and to greatly reduce the cognitive load on individuals for each round of software development. In comparison, participants discussed difficulties concerning long term planning, such as difficulty anticipating long term implications of decisions, difficulty discussing issues that no-one had *really* thought about yet, and difficulty remembering what they had done six months ago when it came time for them to resolve issues at the end of the development process:

> *Now that we are kind of falling out of agile a little bit we are...starting to...not do waterfall but we are kind of starting to forecast what we are going to be working on in the next year and things like that. So it's putting a bit of a strain on development and us I think because, you know, development needs to identify how long it's going to take them to do something and then we need to get requirements. But no one's even thought of it because it's, you know, 12 months from now kind of thing. So we're all kind of figuring out still how we can manage that.*
>
> (T.5.17)

Splitting the development process into iterations was seen to increase energy in the team environment and allow the team to work together more cohesively. Participants related a definite preference finishing one set of tasks as a team, and then being able to put that behind and move on to the next set together. Such activity was related to a sense of completion regarding tasks, and the completion and progression of tasks was seen to give team members something to look forward to as they worked, both as individuals and as a team.

An interesting finding was that the presence of an external goal and deadline *in addition* to iterative delivery was seen to be essential in fostering the kind of 'buzz' associated with high-performance teams:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I think you know, teams that have a sort of...it's not very pro agile thing to say but I think probably one of the differences is where projects where there is a definite deadline, and a definite definite goal of some project that has to be met. A little bit more, you know, just looking a bit more focused when there is some sort of fixed deadline and some definite stuff that needs to be done.

So you find that more motivating?

Yeah, whereas there are some agile projects where there is a burn rate, and you are sort of, you are doing as much as you can every iteration, and you know you are doing the highest priority stuff. But you never...I mean it sort of is what agile is all about, but you know there is no point at which you really sort of finish. You know, you're always adding new features and always doing more stuff. So it's not – there isn't really a sense of completion that you get with a, effectively a less agile thing.

(L.5.18)

It is unclear from this study's data exactly why an end goal above and beyond iterative delivery is so important to team cohesion, but it is perhaps related to feelings of anticipation, and the importance of a holistic understanding of the development process and the product under development.

Finally, consistent iterative delivery was seen as a source of pride, and highly related to the motivation to maintain team standards. It was additionally associated with high levels of trust and security, in that agile team members were well practiced and assured in their ability to work together as a team to produce results:

I don't think that we would have been so consistent in our releases if we had done it otherwise ... The main point is that I feel more secure about what I'm doing. Very strongly. It's also what they tell you the basic reasons behind unit testing, continuous integration. And it's basically that. And I really feel it like that. You really prove after two weeks that this thing is really working, in a more or less stable way.

(X.4.11)

We will do it. We always do it.

(O.2.13)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Process Improvement

Agile practices such as transparency and iterative delivery were seen to improve the ability for agile teams to observe their collective efforts in light of the resulting (or not resulting) software product:

> *But there is a lot of these big organizations that have this myth of – 'Well that's not how we build software.' But when I ask them how they do build software and it's just this chaotic blur of noise that comes at me and you can see them...you can see two people say two different things and look at each other and say 'What?' And you know, they may have worked together for three or four years because they don't have a holistic understanding of how they build things. And I mean, that's one of the biggest differences.*
>
> (X.1.4)

Cohesive agile teams, in particular, were seen to take advantage of each iteration to work to progressively improve the ability of the team as a cohesive software development unit:

> *To some extent it is up to, ah, developers are good at problem solving, and also business analysts are also good at problem solving by, almost facilitation. So helping you find the way to solve your problem. So you've got two skills there that if you work together you can probably say 'Well you know last week's situation kickoff didn't go too well, we almost felt mauled. You know one side of the other might feel mauled at the end of it and so...' You know you might come out with a better iteration kickoff; I mean we have taken some time to get to a balance where the iteration kickoff is not like 5 hours long where everyone is pooped and wants to just go home and you hate iteration kickoff's. To like having – okay it's very easy to estimate but no one is questioning this.*
>
> (L.4.80)

The act of process improvement as a team was shown to be incredibly powerful in improving team relations. Improvements were usually initiated through retrospectives – team meetings that occurred on a regular basis. Such meetings allowed individuals to discuss and resolve problems and concerns, and additionally highlighted the positive aspects of team interactions. The fact that the team as a whole was made aware of, and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

more importantly acted to improve, both team processes and the lives of individuals in the team, was strongly related to increased feelings of appreciation, acceptance, ownership, and involvement in the team environment, seemingly regardless of the actual improvements that were made. It should be noted that several participants discussed the importance of having a good mediator for team retrospectives, and many participants highlighted the importance of having team members who were friendly, open-minded, and flexible in dealing with others.

Process improvement and associated team discussions were seen to foster a deeper understanding of the underlying motivations and intentions held by other members of the software development team, and thus support the development of trust, respect, and improve the capacity for successful collaboration between roles:

> *You know that's, you get a sense of just how to interact with people. And before that I think it was much more separate...the developers knew each other pretty well and how they talked to each other. But the product specialists who sort of defined the direction of the features we talked to them maybe once a week. And so you don't get a sense so much on how they communicate or what they complain about, or, you know. You get a different impression.*
>
> (T.2.21)

> *I mean each time we are working with people we haven't worked with before and this group was a group we hadn't worked with before we have to go through that little bit of 'Yeah you understand what we are doing but you don't really understand it, and so that's you know let's have you watch some of the stuff and we will explain really what we are going to do so that you understand that what we are going to bring to the table.*
>
> (T.4.20)

Thus agile methodologies were seen to increase the ability of team members to understand the roles of others on a personal level rather than in an abstract sense. The unmediated and unfiltered face-to-face environment, in particular, seemed to allow team members to clearly see the effect of their decisions on the project and others in the team,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and to gain a better appreciation of the perspectives, needs, wants, and constraints of others.

Continuous Integration and Automated Testing

Agile methodologies were seen to provide structure in the team environment through the practice of continuous integration and automated testing. Automated testing was seen by a number of developer participants as the most valuable agile practice:

> *Agile or not, I think automated testing incredibly improves, well I guess I'd say 'programmer happiness' as well as product stability.*
>
> (T.2.49)

Well-maintained code in the agile environment allowed them to quickly and easily integrate new features into the software product, with regression test suites ensuring that they were not interfering with another part of the software, or would soon know if they did so. Participants related this practice to enjoyment and pride in the development task, particularly resulting from the newfound ability to be creative, exploratory, or opportunistic in code design and refactoring, and improved ability to be responsive to team member, user, or business needs.

Continuous integration of individual code was seen to ensure that all development efforts were highly focused on the team goal of working software. The team-wide back-stepping and rehashing of efforts when individual code was not continuously integrated into the shared code-base was seen as a major source of annoyance to all involved. The capacity for agile practices to reduce or eliminate this aspect of group coding was related to satisfaction and enjoyment in software development as a team activity; to the point where getting *all* the tests to run was a strong source of pride and motivation in the team environment:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *The team pays a lot of attention on quality so one of the things that agile is introduced here, for example, it's been automated testing. So the team has just gone nuts, I think it's probably the best way of putting it, on automated tests, and we have no other product in the building like it...it's been pretty constant over time and I think the team is justifiably pretty proud of that. And I think this really brought them together.*
>
> (T.1.8)

Continuous integration and testing was thus seen to allow a focus on quality and craftsmanship in a way that was previously difficult to achieve within a team. A major benefit noted by developers in this study was that agile software methodologies increased the chances that their code would be fully integrated into the final software product, and would additionally be able to stand the test of time:

> *Personally I just like to be able to know that something I have worked on, to get that feeling that, 'Hey this works really well' and that users appreciate it, and you know you can tell if have written really good code. You often get a sense that you had time to do it right and it works well and there is not many bugs coming in on it.*
>
> (T.3.36)

This ability, in conjunction with the focus of agile culture on discipline and quality, was seen to be supportive of the rigorous application of agile practices. Team discipline and rigor were further seen to be highly dependent on social pressure in the agile environment:

> *The standards higher; in terms of how the developers hold each other accountable for, you know, somebody checks something and it doesn't have adequate tests; then they will hear about it, from more and more people.*
>
> (T.1.31)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A Note on Communication and Collaboration

This study involved an exploration of the nature of cohesive agile teams in both shared team rooms and in dispersed environments (i.e. the team is in the same building but spread across different areas or floors). Across both types of teams, agile methodologies were seen as valuable in increasing communication and collaboration among team members. Particularly, agile practices were seen to improve team relations in their capacity to reduce social (as opposed to physical) barriers. The social barriers most prominent in this study were: reluctance of team members to waste their time or the time of others; reluctance to interrupt others; and unwillingness to expose or discuss problems. Another reason for lack of necessary communication was simply procrastination while individuals were busy with other tasks.

There were three main ways in which agile methodologies were seen to allow, increase, and even necessitate, communication and collaboration in the team environment despite these barriers – by allowing whole team awareness and feedback, by increasing investment and involvement in the collective endeavor, and by supporting frequent and informal face-to-face interaction.

Whole Team Awareness and Feedback

This research highlighted a subtle requirement for cohesive teams supported by agile environments, which is *constant feedback to the individual that all team members share awareness of team activity and commitment to team goals.* Awareness and feedback elicited from the team *as a whole*, as opposed to through individual or partial team communications, was found to be particularly important. Sharing knowledge and receiving feedback on a team-wide basis was seen to allow a sense of 'common

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

knowledge,' which could then be used as a basis for action that has been 'approved' by the whole team:

> *And its basically having the sort of collective thing of well, agreement that there is a problem and then agreement that there is just a way we can improve things rather than having a sort of whinge and then you know not being able to do...or feeling powerless you know in terms of - oh we have to live in this environment you know.*
>
> (L.4.13)

Daily or weekly team meetings, in particular, in which team progress was assessed, were shown to be an important forum for group action. Team members expressed a strong aversion to showing up to team meetings without having contributed to the team goal:

> *And I think [it's important to have] that daily meeting where you kind of affirm that everybody is on the same team and you have to tell people what you are doing for the good of project – like you can't sit there...and say, 'Well, actually I have just been working on my own thing.' <laughs> Or it's a little harder to do that right, like 'Well I haven't been working at project at all!' <laughs>.*
>
> (T.4.39)

The group forum was also related to the ability of individuals to engage in initiatives that affected the whole team:

> *It's easier for me to ask some of the bigger questions about the product. Then I don't know what other forum we would have. I mean I could talk to the director directly and say 'Well what's happening with this particular contract or what's happening with –? We were talking to a possible customer and what happened with that?' And he could tell me directly what happened, but then no one else would know. Do you know what I mean? I did mention something about, to possibly working in the idea of customer fieldtrips for us to go to customer sites. And you know, I could bring that up with him directly, but it's better to bring it up in a general forum I think because then everybody will see what's going on.*
>
> (T.2.30)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Shared team meetings and environments, therefore, were seen as forums for team members to bring up non-task based items that were of interest to the whole team, such as software architecture, non-functional requirements, and user or customer feedback that might otherwise be 'lost in the gaps' of individual or non-team-wide communication.

Several aspects of the agile environment were seen to amplify the effectiveness of team meetings as a forum for group accountability and action. In particular, the limitation of team expectations to plausible, achievable, and noticeable chunks, such as through stories or iterations, was seen to be important. Cohesive agile teams were further seen to assign relatively limited responsibilities for non-task based items to individuals that were agreed upon by the entire team. After a short period, when things are still fresh in people's minds, teams would then revisit these items, and the progress made towards them or towards over-reaching goals:

> *And that's where the positive thing comes out of it because what happens is you might decide to prioritize the ownership of these actions that you are going to take to further mitigate or improve the bad situations. And so somebody has got to have responsibility for maybe just 2 out of 10 of them you know just however many you know of the...and what happens is that the next stretch is like somebody will go over and say 'Well this, this and this was dealt with is everyone, is everybody satisfied that it was dealt with?'*
>
> (L.4.11)

By maintaining team awareness of individual member activity, and by assigning plausible chunks of action to individuals, agile teams were seen to allow for an extremely high level of social support and accountability in the software development environment. The additional question in the above quote – whether everyone was satisfied with the results of the task assignment – was further seen, when asked in earnest, to increase the probability that a problem would be solved permanently rather than smoothed over temporarily, and to increase individual satisfaction regarding team activity.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Repercussions for bad quality or incomplete work in cohesive agile teams in this study tended to be social rather than punitive. Such social accountability was seen to come from frequent questioning or joking surrounding individual action (or lack of action), from clear expression of need in front of the entire team, or simply from individual expectations of disappointment or disapproval from the rest of the group. Conversely, problems or issues faced by individuals were also presented and explained in the team forum, meaning that reasonable failure to achieve goals was met with understanding and support from the rest of the team as opposed to exasperation or annoyance. Such whole team awareness and acceptance in the agile environment was associated with the increased sense of individual responsibility and self-worth observed in agile team members regarding project activity, above and beyond that provided by support received on an individual basis.

The ability of agile methodologies to support and encourage whole team involvement in regular team forums was seen by participants as extremely valuable:

> *So all of those people came everyday to the Scrums and talked and actually even the marketing person came to these Scrums and talked. So I think what I liked about it was the enthusiasm I guess that everybody had for the project. Having those people show up everyday and having people, you know just really enthusiastic about what we are doing, what we are trying to achieve and all really backing the project.*
>
> (T.4.1)

Having all team members present in agile team meetings or team rooms allows for ease and speed of communication and collaboration. Additional effects were noted throughout this study. Firstly, the presence of all team members at meetings was seen to increase perceptions of the importance of the collective endeavor. Secondly, individuals were seen to appreciate the holistic awareness provided by the agile environment, in that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

they could be assured that there was nothing occurring elsewhere that they were missing or should be aware of. Individuals were also assured that all actions were validated by the group, and that there would be no need for backtracking or rehashing work at a later date. Whole team involvement, therefore, was seen to additionally reinforce individual sense of security and control, and to allow for individual buy-in and investment in the collective endeavor.

The fact that individuals were involved and interested in team communications was seen to strongly affect team relations and individual perceptions. Cohesive agile teams in this study put a high level of importance on ensuring that communications were interesting and relevant to all parties involved:

> *I know that for our group we tried very hard to make sure that what we were saying would be interesting for everybody. Like we didn't bog down things with things that we cared but nobody else cared about and I think...except for a couple of developers <laughs>, most of the people would do that too. They wouldn't say, you know, 'I have just answered an email' or something like that. They would just say those few things that were really kind of generally interesting. And so that kept that momentum going.*
>
> (T.4.4)

> *And I mean, again, it's only the relevant ones for, which you can feature that are involved, so that's nice too because you're not sucking up the time of the whole team to listen to things that they may not be working on.*
>
> (T.5.12)

Cohesive agile teams in this study were seen to support the presence of an interested audience by allowing individuals to opt-in or out of daily communication and collaboration as they saw fit. This was seen not only to speed up communication and increase team momentum, but to assure individuals that everyone present was truly aware of the discussion at hand. Such interest-based participation was therefore seen to maintain certainty of shared awareness as a basis for trust and collective action. In comparison,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

communication in environments where interest was not maintained or assured – such as in long meetings where some team members may not be listening, or group emails where there is no feedback that everyone has in fact read and digested the contents – were seen to stymie team member efforts to move forward as a cohesive whole, and were related to a lack of trust, and lack of team cohesion or 'togetherness:'

> *I was recently working with a non-agile team and I couldn't believe how much I did not feel a part of the team. I didn't understand where the information was coming from - they didn't have scrums. You know, there was some flow of information but it was really hard to figure out where it was...I mean was it email that I wasn't getting? Was it a meeting that I didn't know about? Like there seemed to be this weird way of getting information that I had to try to squirrel and figure out before I could get in on the stuff. But I never really felt I was completely part of the team. And I talked to other people and they didn't either. It was a larger group. So it never really gelled as everyone's working towards the same thing. It was a bunch of people working towards different things.*
>
> (T.4.38)

Opt-in and opt-out behavior in agile teams was discussed as strongly mediated by a heightened social and task awareness in the agile environment. Agile team members interact with each other at least once a day at daily team meetings, and possibly all day in very close quarters if they are in a shared team room or pairing with another team member. Agile environments were thus seen to allow individuals to keep their finger on the pulse of the project and involve themselves when necessary:

> *Like I might be working on my own little bit of the job. But I will kind of be aware of what the guy next to me is doing as well, and what the guys across the other side of the corridors are doing. Because you are right there and you hear the discussions. You are not actually involved in the discussions but you have got an opportunity to be involved in them. If someone starts talking about something and they start talking about something that is related to what you are doing, then you can prick your ears up and you can join in the conversation if necessary. Whereas if they are on a different floor, then you are never aware that that conversation has taken place, So they might be working on something that effects you but you didn't know that. It's good to just...it's much easier to just keep your*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ear to the ground and get a good picture of what is going on when everyone is sitting together.

...

And you don't find that having these conversations going on around you is distracting?

Ah strangely enough no.

(L.3.22)

Awareness of activity in the agile environment often occurred through listening to the rest of the team while visual and cognitive attention were concentrated on other tasks. As noted in the last quote, agile team members often 'prick up their ears' if they thought they heard something interesting, or wrong, or related to their own work.

The agile team environment was also seen to increase the chance that team members would notice the absence of an individual when their input was required for proper team functioning, or otherwise notice the absence of whole team awareness and feedback in the team environment. Agile participants, and particularly members of cohesive agile teams, often discussed behaviors that worked to reduce such feelings of non-communication; such as by drawing team members into conversations, by asking questions and offering information, or by pressuring others to participate in team activities:

There is somebody who recently joined our team...and for a long time he wasn't, for the first two weeks he was not coming to the daily standup meeting. And I distinctly felt like...I didn't know what the heck he was doing. He sat just beside me! And I didn't know what the heck he was working on and what current task he was doing and I did end up, I think I asked him a couple of times you know 'What are you doing?' and 'What are you working on?' and 'How is it going?' Because I kind of knew his general area of expertise but I didn't know what he was doing...then I started bugging him about not coming to the daily standup meetings. So I think so now he comes...So I find that, disconcerting if I don't know what everyone else is doing. In a general sense, you know.

(T.2.39)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Participants revealed a strong interest in general indicators of agitation or excitement amongst other team members, as well as detailed information, such as who was having problems, what the problems entailed, and what was being done to fix them. Awareness of team-wide problems, issues, successes, and progress, was seen to allow for high levels of involvement in the project as a whole, and was related to sense of comfort, sense of control, and feelings of responsibility from individuals towards the team endeavor. The absence of such awareness, on the other hand was associated with insecurity, discomfort, and lack of control in the project environment:

> *I felt uncomfortable at the moment that I was further away from where things that were happening. And happening in the sense that things were happening in the project and I was really in one end of the hallway, three person without a connecting door, and you just didn't hear what was going on. And maybe it was just my, ah, I like to hear when things are going on <laughs>. But yeah, you heard less about things already being solved, or why the build was not running at that moment.*
>
> (X.4.20)

Such awareness and certainty regarding both current project activity, and the validity of the team endeavor, was highly related to the use of information radiators. Information radiators such as burn-down charts, for example, were often seen to allow teams to clearly visualize current project status and what was required to complete goals. Such information radiators were discussed as being exceptionally valuable as a source of motivation, excitement, and team cohesion, particularly after the team had been through a period of uncertainty as to whether they were going to 'make-it' or not:

> *I think basically what the burn-up charts did was giving us a very good, very (detailed?) overview of where we are standing. And giving us a sense of control over the situation. Giving us a sense of the things, where we suddenly saw that 'Yes we can make it. Where there were other things that we already had a feeling that it won't work, but it we couldn't put the finger on it. And the burn chart gave us a way to actually, I don't know how to say – materialize? – this feeling. And to get some very simple way to communicate to management. To*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

point to something and say – 'We have a problem here, and we are not quite sure what to do, and please help us.' And I think that's what saved us.

Saved you from what?

From. Ah, I don't know the saying in English. But in German you would say 'drowning in chaos'...I think we all had this sense of urgency, but everyone was working for himself alone, not as a team. Because we didn't have the time to organize. And the burn chart gave us an opportunity to organize around it...to make a team again. And to give it a common goal. To make the chart look like it will work...now that I think about it, before hand, everyone had his own tasks, his own stories, and he was concentrating on that and was kind of blindsided. And, um, the information radiator was a way to show that there was a team goal that we had to reach together.

(X.3.23-28)

Buy-in to the Collective Endeavor

Investment and involvement in the collective endeavor, such as supported by whole team awareness and feedback in the agile environment, was seen to greatly enhance individual willingness and ability to interact with others. In particular, willingness to ask for help and to give help in the team environment was seen in both agile and non-agile teams to be highly dependent on the dominance of a collective goal over individual goals or tasks. One participant, for example, discusses how helping behaviors are reduced in an environment where personal tasks gain priority over the collective task:

At that point the bugs aren't shared. They don't seem like a shared responsibility to fix. And so I am more reluctant to ask someone to help me on fix a bug because I know that he has his own bugs, as well, to work on. And you know maybe he is less likely to offer help because he has his whole area of bugs as well. And so because they are parceled off to each individual I think it's harder to force someone to help, come and help on your bugs, and likewise share bugs. But when it's at a feature level and like its some larger area of code then I think it's easier to draw someone in and have them share an area of code as opposed to individual bugs. Which is too bad. Because I have some bugs I need help on <laughs>.

(T.2.8)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Focus on a collective goal was seen to reduce the perception of dependencies in the team environment, particularly in comparison to role-based or task-based team environments, where helping was often seen as an inconvenience:

> ***Do you feel when you are doing your work any dependency from the team?***
>
> *Yes. But that is not any different between agile and non-agile projects. You can have a situation where the team is depending on you at any time. The component is not going to work until my bit is done. So even if they finished with their tasks yesterday and I wasn't going to get my bit done until the end of this week, then they are really depending on me to get my bit done. It's actually I think, accentuated on a non-agile team that dependency. Because on an agile team, if my bit didn't look like being finished until the end of the week, then um, then they wouldn't be twiddling their thumbs waiting for me to finish it. They would be like "What can I do to help?' And people still do that on non-agile teams, people are not totally siloed. But it's just like breathing on an agile team. that's just like, that's the normal thing. Whereas on a non-agile team it can be like "Well I'm helping you out here; I'm doing you a favor.' And you are like 'No you are not. We are both doing the same number of hours on this. We are both aiming for the same goal. We are both just doing our jobs.*
>
> (L.3.17)

The sense of not only being 'in the same boat,' but of all 'rowing in the same direction' in agile teams, was seen to greatly reduce stress on the part of individual team members, who no longer had to maintain a reliance on others who were less willing or less able to help them on a regular basis:

> *It's much less stressful because you can rely on other people. It's much more safe, in some kind of way; because you aren't waiting that one day the others will help you.*
>
> (X.3.52)

The presence of a collective goal in agile team interactions was also related to heightened levels of interest from individuals into the activities of others in the project, high levels of sharing and questioning surrounding work tasks, and heightened levels of excitement, involvement, and inspiration in the team environment:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *And you know you get this positive feedback from what you have said, and so you would want to keep talking about it. Like when the marketing guy came over and said 'Okay well I am working on this deal,' and we were like 'Oh tell me about the deal! We think this is really cool.' You know he is going to come back and talk a bit more about the deal and stuff he is working on because he is getting a positive reaction to it. As opposed to 'Oh we don't care about what that is. Tell us when you finally seal the deal.'*
>
> (T.5.4)

Awareness and interest on the part of team members, and on the part of the whole team in particular, was further seen as strong motivating factor in the team environment:

> *I told you about the button that we have to press? Well in fact we also have a website that you can go to and press the button there and get the same impact. And I still like, I think to get up and press the physical button. Because one thing is that it is just more fun. Cool in some kind of way. And the team notices who is pressing the button. And even though when you press the button on the website there is some sound played, and the state of the traffic light changes. It's often something that the rest of the team won't notice as much. But when I am going there to press the button, they just look at me, and they notice the kind of team directions that's missing with the software tools...the interaction with the team, is something that is very important. I feel as part of the team. And that's happening while I walk up.*
>
> (X.3.6)

In this example, a situation where the whole group *notices* individual contribution is compared to a situation where the contribution is made, but is less noticeable to the rest of the team. Cohesive agile environments in this study all involved a high level of awareness of individual contribution to collective goals, usually by way of a highly visible information radiator in a shared team environment. In the example above, the team was using a traffic light that lit up green when a working software build was running, and red when it was broken. Noticeable measures of individual progress included the use of toys that make sound, marks on the team room project board, or simply the evolution of the final software product itself; often it would be something

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

quirky or fun. The important thing seems to be that everyone in the team notices what is happening:

> *And that little dot is like an endorphin rush. It's like, 'Woohoo!' And the team sees the progress, and it's great.*
>
> (O.3.16)

An interesting factor was the frequency of these measures of progress. The events described by participants, such as writing a unit test, or a pair integrating their code into the system, occurred a number of times every day in highly cohesive team environments. There seems to be value, therefore, in *frequent* signs of progress in the team environment. Such indicators were seen to strongly support individual motivation to contribute to team goals:

> *The ritual involved in, having written a test, standing up and going over to the whiteboard, and looking around that, a (keeping?) that everyone see what you are doing. And getting, and making your number of strokes, and getting praise for it from the rest of the team, and everyone else feeling that now they have to, ah, have to keep up writing tests. So that they don't fall behind.*
>
> (X.3.2)

Part of the value of such awareness of individual actions was seen to be in its capacity to strongly affirm team sentiment regarding the project and team environment. Such noticeable indications of progress were seen to allow team members to feel secure in the fact that everyone in the team is working towards the common goal rather than working at odds. While something as simple as everyone showing up to the daily stand-up in the morning was related to an increased sense of involvement and commitment on the part of the team, frequent affirmation of team goals through team discussion and problem resolution was seen to be especially powerful.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Assurance that everyone was 'on board' in this way was associated with increased identification with the team, and appreciation of the team goal as something more valuable and worth investing in than if these affirmations were not present, or if there was dissent or lack of involvement on the part of any individual in the team, for whatever reason. Such regular affirmations of commitment were also highly related to trust, respect, positive feeling in the team environment, and increased willingness to collaborate.

Frequent and Informal Face-to-face Interaction

Detailed knowledge of what other team members were working on, gained through practices such as daily stand-ups, pair programming, and a shared environment, was seen to support a high level of informal and opportunistic collaboration. This was particularly true in teams committed to a collective goal. In task or role-based development environments, on the other hand, collaboration was discussed as being hindered by differing schedules or concerns, and as tending to occur only after an individual had become stumped on a particular problem or task.

One of the main opportunities and instigators for communication in this study was the daily stand-up, where team members were removed from their work to discuss activity as a team. The main periods of collaboration noted by participants, particularly those in dispersed agile teams, were after such team meetings:

> *The daily meetings we have running 15 minutes long where you get around the room and they start discussing the product they are working on. At the end of 15 minutes half of the team is still in there working together on laptops because they have heard about the problems and they want to help each other out. So that team meeting in that room becomes that little sort of common area for a while and then the problems all get solved or there is work items everyone can do*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *individually and then they scatter and disperse back into their cubicles. But it's a start.*
>
> (I.1.16)

Daily stand-ups were also discussed as valuable in their capacity as informal, brief, and non-threatening forums for interaction – well suited to a population characterized by high levels of introversion and detail-oriented behavior:

> *I think mainly that's the programmer characteristic is everyone is really quite introvert. So we try to get more social collaboration happening. But sometimes it's very difficult I find. You really have to make a point of getting collaboration to happen.*
>
> (T.2.2)

> *When a software engineer builds a piece of software they have to have a clear idea of what happens in each case. I mean you go down to a machine it has to understand in very precise terms. So the software engineer will then ask in very precise terms of the business people, which the business people assume that the software engineer is being pedantic.*
>
> (L.1.91)

Frequent face-to-face interaction in agile teams was seen to support awareness of team member state, such as whether they were busy, or stressed, or free. Such awareness of others was related to the ability to evaluate team member receptivity for interaction, and to wait for and quickly grasp a mutually agreeable time to question or collaborate. Thus awareness in the agile environment was seen to not only support opportunistic collaboration, but to allow team members to show respect to others, and to reduce feelings of interruption or annoyance regarding team member interactions. The ability to hear a person's voice and see their demeanor was noted by a number of participants as especially important in relation to team member state; particularly in encouraging awareness and questioning about problems that individuals were reluctant to discuss.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Increased exposure to others in the agile environment was seen to encourage familiarity and understanding between team members, and thus increase the willingness and ability of individuals to initiate communication, be sympathetic to the needs or limitations of others, and communicate effectively:

> *It used to be, so for instance before [agile was adopted]...we would meet with the product specialist once a week and they would tell us what they wanted and what was going wrong and bugs remaining and so you get the overall impression was like they were demanding; they were always complaining and never satisfied. Whereas in you know, with our culture of communication with them, you meet, and yeah you still get the same kind of information that you know if things are missing or they wanted to go in this direction and not that direction. But you get the underlying feeling that they love the product and that just remains as the bedrock. And then you know on the top you know we can deal with the little bits here and there, the individual features. But I think you can get a better sense of just the communication between the various groups.*
>
> (T.2.22)

Such understanding developed through day-to-day interaction was also related to individual feelings of acceptance and belonging in the software development environment; particularly when compared to environments where interactions between team members tended to be infrequent or on a need-only basis. Informal communication and familiarity were further seen to allow team members to discuss vulnerabilities or problems in both their own work, and in the work of others:

> *Well, just that because I see them so much more often, and they see me; because we don't sit together, you know. In the beginning when we working on project, and even for a while before we were agile, you would see someone and go over and say 'Do you need any, am I blocking you on anything?' And they would say 'No.' You know, because they don't really know you well though so they wouldn't have a lot...there wouldn't be a lot more discussion other than 'No, I don't need anything from you right now', or, 'Yes, I need such and such a design.' But now that we see each other everyday, and a few times a day it's like 'Oh, how was your weekend?' And, or coming out and saying, 'Well, I don't think this design isn't going to work because such and such.' You know, you are less afraid to speak your mind because you have more of a relationship with people that you work with.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(T.5.26)

Constant and shared dialog as encouraged in the agile environment was also discussed as resulting in well thought out, elegant, and mutually agreeable solutions; more so than if one person had come up with a solution on their own. Particularly, sharing of activity and opinions among team members was seen to allow individuals to validate, enhance, and refine ideas *in relation to team and business goals*. Pair programming was seen as especially valuable in this respect:

> *I think if someone pair programs with you it's more a case of 'Are you going in the right direction?' 'Is this, you have kind of made the step but I think you have a longer term view and does that really mesh with the way its going to fully develop in later versions.' So that's more, it's more collaborative.*
>
> (T.2.12)

> *Well I have been convinced for a while that the main productivity benefit of pairing is that the other person slaps you on the head and tells you 'We don't need that now!' Especially with clever people. Clever coders go of and do clever things, but 'No we don't need it!'*
>
> (L.2.17)

Constant dialog in the team environment was also seen to support a level of *storytelling*, which was a common theme coming from discussions with participants. Sharing of knowledge and ideas through stories and experiences was seen as particularly valuable to members of software development teams trying to convey complex thoughts or insights. Aspects of storytelling can further be seen in the project environment. Agile team members are immersed in project activity, and were seen to gain knowledge of the project and product through frequent updates and observation of progressive events. Such progressive awareness and understanding was related to increased motivation and excitement surrounding project activity, as well as to greater acceptance of requirements coming from other group members. This can be held in comparison environments where

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

one role or group creates a document containing their understanding, which is then passed off to another role or group who must digest this knowledge at once and without the benefit of contextual information.

A number of participants in this study associated the agile environment with a very relaxed, friendly, and social atmosphere, which further smoothed the development process:

> *I was here for about a year before we started with agile and I mean we didn't do these daily meetings and things like that and even just something like that, I have seen a big difference with that. You know, just because we are more involved and we are all sort of thrown into the same room more often, which is going to force us to talk and communicate and find out what each other's working on and stuff like that, so that is definitely a benefit...because we see each other more often, we're very, I think we're a lot more friendly with each other than we would have been before and there is always lots of, you know, joking back and forth and everyone is very, I would say it was a very tight knit group.*
>
> (T.5.16)

It should be noted that it is unclear from this study how much of the social atmosphere discussed was due to agile practices and how much was due to the particular personalities in the teams under discussion. While agile environments were seen to better allow for such naturally occurring social awareness and collaboration, a number of agile teams were discussed that did not engage in such social activity.

Still, the ability to regularly request and share opinions in the agile environment, as well as the ability to share successes and progress with others, was highly related by participants to positive feelings and excitement regarding both individual tasks and team activity. Such social engagement, and particularly the social energy and momentum found in cohesive agile teams, was noted by a number of participants as particularly enjoyable when compared to development environments where they had worked in relative isolation. Female developers were seen to be especially appreciative of the agile

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

environment in this respect. While other participants were less enthusiastic about a highly social team environment, all participants were seen to appreciate the value of agile collaboration in supporting the team-based development process.

Overall, team members in agile environments were seen to be more willing, or in some instances forced, to confront difficult issues that would otherwise be avoided:

> *I think that open and honest communication is a big part of agile culture. And that, and respecting individuals. So I guess could have happened without agile, certainly. But I'm not sure that it would have for us. And another thing certainly is the close collaboration that happens with agile working. Ah, a great deal of help for trust.*
>
> ***What were the barriers that were overcome? What do you think that it might not have happened for you if it hadn't been for that push?***
>
> *For one thing I think that we wouldn't have had such a strong need for it...If there hadn't been such close collaboration, it might have been; like for example, when I first joined the company, everyone was working for himself, basically, um...it would probably just would have said, 'Okay, that didn't went well but it's not that important. We can work. And who cares? Huh?' And having to work closely together simply created the need for solving these problems. And of course, also, helped in solving the problem, because we knew each other much better. So there are also a much stronger personal need for solving this problem. Because it is causing much more stress for you, and you feel much ah...<german word>. It's just no fun when you have to work with someone when you feel that there's some tension.*
>
> (X.3.73)

Participants in cohesive agile teams placed a high value, therefore, on working past or resolving personal issues that disrupted team communications. The act of confronting and resolving problems, as well as expressions of goodwill towards maintaining good relationships, were seen as reflective of the high value placed on team communication and collaboration by agile methodologies and culture. Such activities were related to increased trust, respect, comfort, and feelings of closeness in the team environment:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *Also, it has done a lot for our communication style. So that, recently...I don't remember what it was about that...we had some discussion about, it must have been some minor thing. So we just discussed and we settled the issue, and when it was lunch time before the coworker was just on his way out to lunch. And turned, and came to me and said, 'What I said in that discussion. It wasn't meant that way. You might have understood it in that and that way.' And I was 'Okay, no it was all good, I didn't misunderstood, but thank you for telling me.' And...we had, it was a great feeling of trust between us, because we could speak openly with each other.*
>
> (X.3.71)

A final aspect of the agile environment supported by frequent face-to-face interaction was mentoring. Agile methodologies place high value on the distribution of knowledge across a team, and thus provide a high level of support for mentoring and learning initiatives. Such activities were seen to increase individual willingness to collaborate, learn, and help others, and were noted as one of the major points of personal satisfaction associated with the agile methodologies. Pairing was seen by many participants as a particularly valuable means of distributing knowledge, increasing awareness across individuals in the team, and enhancing enjoyment and satisfaction in development tasks.

A notable finding was that although the personal value of learning was often discussed, the opportunity to share knowledge and mentor others was often seen as more valuable, and was related to feelings of importance, self-worth, and pride. Helping others was noted as *the* main motivator by a number of participants, with their main source of personal enjoyment and inspiration coming from seeing their team members learn and grow:

> *I get most of my joy from seeing other people...seeing them improve, you know; seeing them get better. And the process, and their work, and the way they are developing. It's great.*
>
> (O.2.4)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Self Regulating Teams

Individuals in agile environments were seen to be much more likely to hold an awareness of how they worked together collaboratively as a team; this included a keen awareness of the detailed interactions and dependencies between team members in the project environment, as well as a better understanding of each team role in relation to other roles in the software development process. Such awareness and understanding in the agile environment, particularly regarding the skills and capabilities of other members in the team, was seen to increase the chances of an environment of equal partnership involving non-hierarchical shared leadership and responsibilities, and thus self-regulating teams. The individual benefits resulting from such environments were clear in this study, including feelings of empowerment and motivation, a strongly heightened sense of control and personal freedom, increased initiative and investment into both the software project and the team, and greater satisfaction and enjoyment in work.

Agile, and particularly self-regulating teams were associated with increased discussion and questioning regarding team-wide project activity, and related increases in agreement or acceptance surrounding the eventual resolution or solution resulting from discussion. Individuals in agile teams that allowed a level of self-regulation were further seen to feel more responsible for the well-being of the project as a whole, to engage in thought or action regarding occurrences outside of their specified roles, and to otherwise support collective activity:

> ***I think its empowered a lot of people to help and want to help one another whereas before you didn't have to go an extra mile because your authority was only this big and when you have done that, you are complete, you didn't have to go above and beyond to be measured as a good or effective employee.***
>
> (I.1.15)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Team-based software development was seen to be well suited to the use of self-regulating teams. Participants often discussed a wide range of possible solutions to a given problem within project constraints, and the resources to be considered as including all the knowledge and experiences of the individuals in team and the combinations between them. Agile environments were seen to increase team awareness of the knowledge, previous experiences, and current activity of team members, particularly through face-to-face communication, which allowed non-task based knowledge transfer through activities such as storytelling and mentoring.

The social network of information developed and maintained in agile software development teams through close and frequent interaction, was seen to reduce information overload, as well as the pressure to 'know everything,' and seemed vital in ensuring that the right people were in the right place working on the right task. Participants working in self-regulating teams discussed a resulting increase in their satisfaction in work, and their ability to work effectively as a team to produce high quality software.

Self-regulation in an environment of whole team awareness was also seen to support interest-based participation, and to increase the likelihood that there would be a good fit between team action and the needs and capabilities of individuals within the team. Absence of self-regulation, on the other hand, was related by participants to annoyance or resentment when one individual received tasks that another individual felt that they could do better, or frustration and struggle when one individual received tasks that were beyond their capabilities.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Awareness of the whole team, as provided by an agile environment, was seen to be essential in allowing responsible activity in self-regulating teams. Team members interviewed in this study showed an increased willingness to take responsibility for team tasks when they knew what needed to be done, knew the capabilities of others in the team, and were aware what other people were doing. They discussed quite happily picking up even the most tedious task and doing it when it became clear that no one else was available. Lack of awareness of the whole team increased the chances that tasks or issues would fall through the gaps and not get completed or resolved – not necessarily because no-one was aware of them, but because no one would step up to take responsibility of them.

> *...Not starting anything new but finishing the stuff that they had. And by having it up here on the board, then everyone knew that it was sort of fair, you know. They knew that things needed finishing, and they knew that, you know, that the work had to be allocated to someone. So someone getting lost by...I mean it's very typical of developers, but no-one really wants to do all the finishing, you know, tidying up stuff. They all want to do the new stuff.*
>
> *So it's just bringing a little bit of, ah, sort of visibility for the things that needed finishing for the stories to get done. And in a way that was sort of fair to everyone. You know I wasn't sort of picking on anyone. It is up to everyone to work out what they were going to do next, but they just understood that...someone in the team...had to take those nasty – chasing up someone to find out what the answer to some question was, to get something done – tasks.*
>
> (L.5.28)

Areas for Future Study

This was an exploratory study into agile practices based on core data from 22 participants. Further research, particularly observational study, of agile teams would be required to confirm, refine, or deny findings. This research also highlighted a number of topics and issues worthy of future study, and it is hoped that it will encourage further empirical investigation into human activity within software development methodologies.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

While research was focused on the positive effects of agile methodologies, the study's data highlighted a number of pitfalls and challenges associated with agile software development. The capacity for agile practices to support cohesive teamwork was seen to be highly dependent on how the practices were implemented and in what context. Further study into different configurations of agile practices and management styles, and the resultant and underlying effects on individuals and teams would be valuable in understanding the critical success factors involved in the implementation of agile methodologies. This study also focused primarily on extreme programming as a basis for agile interaction; studies of other agile methodologies such as Scrum, alone or in comparison, would be beneficial.

The close collaboration in the agile team environment was further shown to be particularly draining on some individuals, to the point where they were left exhausted at the end of the day. Even more than in non-agile software development teams, the need for individuals to be 'on' and 'in' the entire day was seen to have additional effects such as the lack of privacy and down time to make personal calls and run errands. Future study into these phenomena, and efforts to instantiate practices to reduce such stress on individuals in agile teams would be worthwhile. A number of participants also mentioned the capacity of agile to reduce the time before 'burnout,' both related to increased contact collaboration with the same team member's everyday, or to increased immersion in the same project activities. Further study into such effects, as well as exploration of the effects and value of turnover on agile teams seems to be a valuable area of study in this respect.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Agile methodologies were also seen to require a very particular style of team member interaction, particularly involving increased flexibility and collaboration. Participants discussed many instances where individuals were unable to operate in an agile team environment, or where the 'wrong' personnel had debilitating effects on the functioning of the entire team. Practitioners experienced in traditional software development methodologies, in particular, were seen to find it hard integrating into agile teams. Future study into working style, personality, gender, and team member interactions would be valuable in increasing understanding of the requirements for individuals working in agile teams, and to determine ways to ease difficulties faced when working in an agile environment.

A related area for future study would involve the transition into agile methodologies. Further exploration of the difficulties and stress faced by team members trying to 'be agile,' and barriers to agile adoption coming from individual, team, and organizational dynamics would increase understanding of how best to implement agile practices, particularly in contexts that do not fully support agile team activity. Conversely, another common experience was difficulty faced by individuals working in *non*-agile contexts after being part of an agile team. Study into what this entails and ways to alleviate the stress of such transitions on individuals also seems useful.

A comparative study of agile and waterfall development methods, and the relative effect that each has on team dynamics and effective software development is also likely to yield interesting results. Particularly the presence of, and activity surrounding: crisis or tension; power and hierarchical relationships; self-regulating team behavior; formality and informality in team environment and communications; innovation and creativity;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

physical space and environment; social interaction and activity; team cohesion; and team flow and/or 'buzz.'

This study also involved a prevalence of participant difficulties in instantiating agile in inherently non-agile contexts. Study into the interfaces and dynamics between agile teams and the outside organization, other teams, and individuals or roles who are not fully dedicated to the team would be valuable in discovering how to alleviate such difficulties. Related to this was the tendency for individuals to engage in 'agile idealism;' in general, but especially in environments ill suited to agile interactions, this was often seen to result in negative effects in the team environment. Further study into such behavior, and ways to reduce it, seems valid.

Other phenomena that was amplified due to the unique, and relatively enclosed nature of agile teams that were highlighted by this study included the tendency for agile teams to become overly differentiated or isolated from the rest of the organization, and the tendency for agile teams to become overly homogeneous over time. A related factor involved a tendency for agile teams to become overly attached to rituals or artifacts that had lived past their usefulness. Further study could involve such effects and ways to reduce associated negative outcomes.

Other areas for future study revealed by this research involve the implementation of agile software development methodologies outside of the context that they were originally developed, which is in small face-to-face developer teams. Interviews with non-developer roles in particular, revealed the developer-centric nature of agile methodologies. Conventional agile practices seem specifically tailored to increase the capacity of developers to collaborate and to ease development of quality software. Other

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

roles in the software development process were seen to have a comparatively tough time in dealing with the agile methodologies, as well as with development teams that employ agile methodologies.

Some roles, such as business analysts or 'customers,' were seen to suffer from increased workload and responsibility, while others – quality assurance specialists and user interaction specialist in particular – were often seen to be under-appreciated and under-involved in agile teams that were largely centered around developer activity. Participants in teams who had adapted agile to allow for increased involvement of non-developer roles, however, were highly enthusiastic about the benefits of involvement of such roles, and the synergy that resulted from developer and non-developer collaboration. Research into value and the integration of non-developer roles in agile methodologies, particularly into the individual and team dynamics involved in increasing collaboration and understanding between roles, seems to be a particularly valuable area for future study.

Interviews also revealed teams that had both difficulty and success in implementing agile methodologies in large or distributed teams. This study highlighted a number of important factors involved in agile communication and collaboration, including: whole team awareness; frequent affirmation of team commitment to shared goals; constant background awareness of team activity and project status; the presence of audience for action and interaction; social pressure and accountability; familiarity and understanding; and the importance of social energy and voice in agile interactions.

Further study of such factors, and exploration of how to reproduce such effects outside of an intimate and shared face-to-face context would be highly valuable in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

allowing the excitement and involvement associated with agile teams in non-traditional contexts. This is another area where a comparative study between traditional and agile contexts seems valuable, particularly in exploring the effects of the use of software development documentation as opposed to face-to-face communication and knowledge transfer in software development teams. Studies into the effects of frequent and less frequent interaction with both project information and team members seems particularly valuable in light of the findings in this study.

This research also revealed a number of interesting phenomena associated with particular agile practices. Information radiators were shown to be especially valuable in supporting engagement, awareness, and collaboration in the team environment. Such artifacts were highly associated with team cohesion and motivation, particularly in allowing the visibility of team goals and the activity needed to reach them, and in providing frequent and immediate feedback to individual and team activity. Comparative study of effective and non-effective information radiator use and resulting effects on teams and individuals, as well as study of the lifespan of information radiators in relation to teams and projects, would likely yield some interesting results.

The estimation and planning process also seems a valuable area for future study in that it involves complex give and take negotiation between roles in software development teams. Further study of the dynamics of such interactions would be useful, particularly in how to instantiate the value of non-developer roles in the planning process. Such study could provide a better understanding of the generation of group agreement in software development teams, and of how to reduce inherent tension and conflict existing between software development roles.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Iterative delivery was shown to have a strong effect on both individual and team activity. Future study into the effects of such delivery on individual perceptions of the importance of team goals, software development team dynamics and effectiveness within condensed periods of time, and the effects of the presence of an end-goal as opposed to continuous iterative delivery would be valuable. The presence and effect of process improvement in teams also seems a useful area for future study, particularly the use of retrospectives in team building and instantiating change.

A practice that was seen to be highly valuable to participants in this study was pair programming. Particularly, qualitative exploration through interviews revealed a number of interesting effects related to coding and design tasks. These included: the ability for pairs to increase the 'head space' applied to a particular design task; the ability for pairs to increase the focus on team goals as opposed to the coding activity itself; differences in the level of abstraction and critique applied in face-to-face interaction during the coding activity as opposed to after-the-fact code review; and the ability for pair programming to increase the 'elegance' of software design. The development of shared code is a particularly interesting area for study, including questions such as: how is code constructed as a team? And, what are the critical success factors for creating well-structured code with others? Pair programming was also seen to have *very* strong effects on team dynamics and awareness of others in the team environment, and further study into these effects would likely reveal some valuable results. Related areas of study that could benefit from such examination of agile teams would also include topics such as collective memory and consciousness, situated memory, distributed cognition, mental models, knowledge management and transfer, and communities of practice.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finally, deeper exploration of agile culture and values and their effects in the software development environment seems valuable; particularly in exploring the differences in how team members consider and uphold these values in agile and non-agile environments.

Conclusion of Results

Overall, agile practices were seen to increase the ability for team members to work together with others in an environment of constant feedback and progress. Such an environment was seen as truly motivating, and agile team members were more likely to exhibit a strong sense of pride, purpose, and excitement in work:

> *What I see as the motivation is the challenge of creating something quite phenomenal, with other people.*
>
> (X.2.73)

Cohesive teamwork was seen to be dependent on the ability of teams to maintain a high level of responsiveness and synergy between team members within a tightly focused scope of activity. Cohesive activity in teams was also seen to be dependent on individual feelings of security, control and involvement regarding the process used to create software, the product under development, and especially, the team on which each team member must rely. Agile methodologies were seen to support such feelings in a number of ways.

The agile plan and the process used to create it were seen as essentially stable structures providing a level of team certainty and security about which disciplined group activity could occur. Everyone is aware of it. Everyone agrees on it. Everyone knows it is correct. Everyone can invest effort in such a plan in the certainty that they are adding

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

value and acting as part of the whole. The chunking of activity into stories and iterations was seen to create manageable chunks of software development. This was seen to allow holistic visions of both team process and the product under development, and for feelings of responsibility and ownership over process and product. Regular iterative delivery was also seen to increase the immediacy and importance of working together as a team, and to increase enjoyment, motivation, trust, and respect in the team environment.

Under the umbrella of the agile plan, team member activity was relatively unmediated by process or documentation constraints, and teams were often seen to self regulate. A shared goal and the close-knit agile environment was seen to allow familiarity, understanding, and acceptance in teams, and thus increase the likelihood that team members would share both their problems and successes with others. Frequent, informal, and interest-based communications in the agile environment were seen to produce coherent plans for action that were well suited to team capabilities, and the agile environment was seen to provide a high level of social accountability surrounding their execution.

Feedback was seen to be especially valuable in that it allowed individuals and teams to *regularly* gain positive input and recognition regarding the value of their actions to others; both within the team, and in the larger context of customers and end users. The centralized nature of agile team activity and communication, supported through practices such as daily-stand-ups, was also essential in allowing team members to maintain an awareness of the state of project and team environments. This was seen to result in increased feelings of security and control, and to allow for high levels of investment and action *as a team.* In comparison, long-term planning and implementation efforts and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

piecemeal division of work were seen by participants as understandable, but ultimately unexciting, uninspiring, and unmanageable.

> *You feel like you get things done. And you feel like, you know, you've ticked things off and you feel like not only have you done it but the business says you have done it, you know. You have met the acceptance criteria that was suggested in the beginning and you talk to your business. Rather than just delivering something and they go 'Mmm that's not what I want now' and that sort of thing. So you feel a bit of a buzz, and there is also a buzz about you know when it goes live, but that's of every project, but it's quite nice to sort of see it go quickly you know. Also I think it's, for us it's like, you know, being in the position of learning, and then being in the position of equal partnership and then being in the position of mentoring. You know in different areas. And it's nice to feel that the you're constantly learning something or you are constantly coaching someone else, you know. And also that feeling of communication between the whole, the mini business really. So that's also nice as well. I think that there is a definite buzz to that which is quite hard when you leave <laughs>*
>
> (L.4.69)

Discussion of Research Method

Overall, the grounded theory process was found to be extremely valuable in developing an understanding of experiences in agile software development teams. In particular, the line by line coding highlighted aspects of experience that would not have been noted or considered when reading transcripts on a higher level, such as sentence by sentence. Grounded theory was found to be especially useful in that it provided a means for comparison of commonalities in experiences across a diverse range of situations in the software development context.

One of the difficulties faced in this study was the variety of organizations, software development domains, projects, team composition, and individual roles involved in discussions with interview participants. Obtaining a clear understanding of the variables affecting each project, and comparing experiences in different projects to each

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

other was therefore quite difficult. The data collected in this study was extremely diverse, and unknown differences between participant experiences may have confounded results.

The nature of the research also provided some difficulty in that many participants were incapable of, or indisposed towards articulating their feelings and personal experiences in the team environment. Results of this study, therefore, involve a high level of inference regarding participant experiences based on language and intonation used during individual interviews, and on the general sentiments felt across interviews and through participation in the agile community. As such, study results are especially subject to researcher bias.

Part of the data in this study may also have been confounded by the fact that five of the interview participants were from the same organization, which involved a highly innovative product development environment. Due to the small size of the study sample, the experiences of these team members played a large role in theory development. Efforts were made, however, to avoid making generalizations to other software development domains and environments based on the experiences of these team members, and to confirm the experiences of these individuals with those of other study participants.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Discussion

This research highlighted the importance of studying socio-psychological phenomena occurring during software development; particularly, the scarcity of academic research into subjective experience was seen to leave a gap in our understanding of positive psychological phenomena, such as motivation and excitement, occurring in work teams. A conceptual framework depicting agile teams as complex adaptive socio-technical systems highlighted the importance of observing the context of individual work and teamwork, of viewing teams as holistic entities, and of exploring the mediating effects of technology on human activity. Further analysis of human agency within such a framework highlighted the value of observing interactions between individuals and the group as a whole. Qualitative grounded theory was seen to be particularly valuable in exploring the dynamics surrounding teamwork as mediated by agile methodologies, and in working towards a holistic understanding of team activity that crosses the boundaries of traditional disciplinary paradigms.

Collective Culture, Self-efficacy, and Feedback Loops

A collective culture of interaction was found to be essential to individual motivation and excitement regarding the task of team-based software development. Collective team functioning, where each individual was aware of and invested in the activity of the team as a whole, was seen to be essential to feelings of personal security and control in the software development environment. This is understandable considering the nature of software development as an unstructured, complex, creative, and *social* design and problem solving activity (see Warr & O'Neill, 2005). The solution space for any given software problem can be seen to expand or contract based on the abilities of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

individuals in the project team, where any number of software solutions could plausibly satisfy a given problem. Software development outcomes seem further dependent, not only on individual knowledge and decision-making, but on the resolution of interdependences between team members, the synergy resulting from team-wide discussion and collaboration, and the ability for all team members to share a common vision for the software to be developed.

Such activity and coordination can be differentiated from 'knowledge work' or information processing (Drucker, 1993; Morhrman, Mohrman, & Cohen, 1995), as well as from mechanized production work structured by physical technology. Cohesive software development teams in this study were set apart not by their ability to "leverage specialist knowledge" or "use more fully the human resources of today's organizations" (as discussed in Muthusamy, Wheeler, & Simmons, 2005), but by their ability to get all members of the development team to work closely together towards a common goal.

Agile methodologies were seen to allow, support, and even require the development of a collective culture over time. The instantiation of such a culture was seen to be highly dependent on feedback and feedforward mechanisms in the agile system, with agile practices supporting heightened team member awareness of collective tasks, goals, and progress. A study by Eby and Dobbins (1997), for example, relates preference towards collective behavior in groups to positive past experience working in teams, self-efficacy for teamwork, and the need for social approval. Positive past experience was seen to be supported by agile planning and iterative delivery, which allowed teams to develop a history of successes, thus increasing the likelihood of team member preference for collective activity (see also Breer & Locke, 1965). Self-efficacy

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for teamwork was seen to be particularly well maintained by agile methodologies, thus also supporting team member preference for teamwork over individual work (see Paulhus, 1983; Campion, Medsker, & Higgs, 1993).

Eby and Dobbins discuss team self-efficacy with regards to efficacy expectations and locus of control (Bandura, 1989, 1991, 1992). Efficacy expectations involve perceived self-efficacy regarding effort or skill expended in effecting change within a specific context. The related but distinct concept of locus of causality or control, on the other hand, involves the amount of controllability or modifiability of one's environment.

Agile practices such as daily team meetings and continuous integration and testing, which focus daily activity on the collective goal of working software, and which provide constant feedback on the validity of individual actions in relation to team goals, were seen to greatly increase perceptions of self-efficacy and control in the team environment. The negotiation of a flexible plan well suited to team capabilities was further seen to increase individual perceptions of team-wide self-efficacy and control, along with other factors such as positive feedback in the agile social and development environment (see Bandura, 1986), and detailed and holistic awareness and involvement in project activity, particularly by way of information radiators and the evolutionary development of a working software product. In contrast, software development environments where those conducting development tasks were relatively uninvolved in the development of a detailed project plan, or where team members were not clued into the day-to-day activities of others, were related to high levels of discomfort, dissatisfaction, and the absence of perceived team self-efficacy and control, seemingly regardless of the actual state of the project.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

While this study was not focused on measures of performance, it should be noted that the presence of self-efficacy in the team environment indicates a socio-psychological environment of high-performance. It is generally accepted, for example, that performance in both physical and academic tasks is enhanced by feelings of self-efficacy (Baron & Byrne, 2000). Such effects can be physiological (Bandura, Cioffi, Taylor, & Brouillard, 1988), or socio-psychological. Increased self-efficacy and related levels of self-esteem, for example, have been related in medical practice to reduced feelings of stress and to reduction of redundant or unnecessary procedures leading to increased costs (Huang, 1998). Indeed, participants in this study often discussed improved task effectiveness resulting from a heightened awareness of others in the agile environment.

Karau and Williams' (1993) Collective Effort Model (CEM) adds to such findings. According to this model, individuals will work hard on a given task only to the extent that a) they believe that their hard work will lead to better performance, b) they believe that this performance will be recognized and rewarded, and c) the rewards are ones that they value and desire (Baron & Byrne, 2000). In other words "individuals working alone will exert effort only to the extent that they perceive direct links between hard work and the outcomes they want" (Baron & Byrne, 2000 p.492). Karau and Williams further discuss how such links are likely reduced for individuals working in groups. The presence of factors outside of individual control, for example, such as the amount of effort put forth by others, reduce the linkage between individual effort and performance.

Aspects of agile teams, such as transparency, highly focused iterative delivery, and noticeable measures of progress can therefore be seen to increase perceived linkages

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

between day-to-day effort and valued collective goals, such as delivery. Agile environments thus provide motivation for individuals to work harder towards team goals when compared to environments where team members are less aware of the activity of others, or where it is less clear how the team is working together to produce results. An interesting finding in this study was the importance of *frequent* feedback regarding team activity. Informational or pragmatic awareness of team activity and plans, it seems, is often not sufficient for perceptions of team self-efficacy and control. Weekly, daily, and even hourly feedback and affirmation of team activity as provided by agile methods was seen in this study to be essential in providing the security and motivation for team members to buy into and put effort towards collective goals.

Non-agile environments, on the other hand, were associated in this study with downward spirals of self-efficacy, where lack of awareness of team activity leads to reduced perceptions of team self-efficacy, causing individuals to exert less effort towards team goals, resulting in reduced team self-efficacy. In such environments, team members seemed to shift the focus of their efforts to non-collective activities over which they had more control, such as role-based tasks, thus undermining whole system efforts through optimization of subsystem goals. The software development environment in particular, characterized by extremely high levels of complexity and uncertainty, is therefore seen to be ill suited to specifications-based component architecture of software systems in the Taylorist (Taylor, 1947) tradition, primarily because of the emotional needs of individuals working in software development teams.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Perception and Intent

Eby and Dobbins (1997) support the observation of collectivist orientation free of context, where individuals either hold or do not hold a disposition for cooperation and group work towards collective goals. This research suggests, however, that disposition towards collective activity can be fostered in the team environment over time. Results further indicate that disposition towards collective or collaborative activities is often trumped by other factors in the software development environment; particularly, the presence of and team progress towards a meaningful collective goal.

The need for social approval, for example, was seen by Eby and Dobbins to support collectivist orientation, and social and collaborative activity was shown to factor significantly in cohesive agile teamwork. While some participants noted a close-knit social environment as one of the most valuable aspects of agile teamwork, however, other participants were relatively unimpressed by social activity. It is notable, therefore, that such team members were still seen to value and engage in collective activity in software development. Agile methods were thus seen to provide motivation for even non-social team members to collaborate, in that the software constructed as a team is much more complex and technically challenging in structure than what any individual can create on his or her own. Again, pragmatic understanding of such a structure that would exist in the distant future was not found to be inspiring. Constant awareness and feedback regarding the evolution of a working software product, on the other hand, was discussed by participants as deeply satisfying, and was seen to sustain high levels of motivation from week to week.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The application of psychological research to agile team practice was seen to be affected by the presence of team-wide buy-in and commitment to collective goals. While factors such as propinquity (Festinger, Schachter, & Back, 1950) or reciprocity (Thorn & Connolly, 1987), for example, have been noted as important factors in communication and collaboration between team members (see also Goodman & Olivera, 1998), individuals in cohesive agile teams were seen to work to overcome such barriers to team interaction in order to reach team goals.

The importance of a final end goal in supporting high levels of motivation and excitement in software development teams as revealed by this study speaks to the importance of intent in human systems, where system activity is purposeful rather than mechanical or deterministic (Ackoff, 1999; Gharajedaghi, 1999). Application of psychological research in practice was seen to be additionally dependent on individual perceptions of team activity, which were in turn dependent on individual end-goals. Thus research postulating that observation of past team successes will support a preference for collective behavior, is in practice highly dependent on individual perceptions of 'success.'

A success from an organizational perspective, for example, is not necessarily related to personal feelings of success. Such disparities between individual and organizational goals were seen in this study to greatly undermine individual motivation and satisfaction in work, and can be seen as damaging to the organization in the long term (see Demarco & Lister, 1999; Humphrey, 2001). Agile methodologies were seen as valuable therefore, in their capacity to allow individuals to self-regulate within clear team and organizational constraints. More than this, agile methodologies were seen to structure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and mediate awareness and perception of self, of the group, and of the interactions between the two in a way particularly well suited to the instantiation of purposeful and collective teams.

Prioritization and chunking in the agile planning process, for example, was seen to focus team awareness, with plausible and comprehensible plans for team action supporting collective thought and activity. The externalization of goals and progress, such as through discussion and dialog between team members, and through highly visible and interactive physical artifacts or information radiators (Cockburn, 2002), was seen to further support heightened awareness and coordination between team members. Information radiators were exceptionally effective in highlighting individual actions in light of team goals. The act of standing up and making a check mark on a clearly visible information radiator in a shared team room, for example, was highly related to feelings of team 'togetherness.' Thus agile team environments can be seen to constantly reaffirm individual activity and intent in relation to team goals, heightening perceptions of importance of both individual and team activity.

Social Identity

While recent research regarding self-regulating work teams has centered around organizational, rather than social psychology (Ilgen et al., 2005; Levine & Moreland, 1990; Sanna & Parks, 1997), the social perspective was essential in describing activity in agile software development teams. Social identity theory (Tajfel, Billig, Bundy, & Flament, 1971; Tajfel & Turner, 1979), in particular, explores individual psychology within a social context, and offers a number of valuable insights into agile teamwork.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In social identity theory, individuals have several 'social identities' corresponding to perceived membership in social groups (Hogg & Vaughan, 2002). In contrast, personal identity stems from self-knowledge of unique personality traits and interpersonal relationships. The salience of a social identity in an individual at any one time is seen to vary based on social context (Turner, Hogg, Oakes, Reicher, & Wetherell, 1987). Certain contexts will therefore increase the prevalence of social identity as opposed to personal identity, and vice versa. In sum, social identities are seen as an essential aspect of a person's self-concept, and differing social contexts will cause a person to think and feel, as well as act differently.

Agile methods were seen in this study to heighten the presence, value, and importance of project team identity as opposed to individual or role-based identity. Constant immersion and engagement with the rest of the team, for example, and the development of rituals surrounding team activity, were seen as strongly supportive of the development and presence of a shared identity. Agile teams engaging in self-regulation and learning initiatives were also related to reduction in identification with software development roles; roles which are likely internalized and linked to key aspects of an individuals' self-perceptions and self-concept (Baron & Byrne, 2000). Thus individuals were seen to identify more strongly with the project team, allowing for improved communication and reduction in the stress associated with inherent role conflict (see Williams & Karau, 1991).

An idea closely related to social identity theory is the notion of social comparison (Festinger, 1954). The theory of social comparison holds that positive self-concept is fundamental to psychological functioning, and is established through evaluation of self as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

compared to similar others. Thus positive self-image and self-esteem can be gained through comparison to other group members, and through identification with a group holding a high amount of prestige. Findings in this study were consistent with this theory, with members of cohesive agile teams noting positive feelings in association with recognition of their abilities within the team and pride in project group membership.

While Tajfel and Turner (1979) discuss how individuals and groups gain prestige through in-group out-group comparisons, however, prestige-based comparisons between agile and non-agile teams in this study were relatively rare. Teams in this study were more likely to compare their state and progress to expectations and reactions of users and business to versions of the software product. Teams were additionally seen to compare their activity to an 'agile ideal' that was seemingly based on editions of *Extreme Programming Explained: Embrace Change*, by Kent Beck (2000, 2004). By providing focus for teams in the form of working software and optimal agile process, agile methodologies can therefore be seen to mediate social processes such as inter-group comparison.

The Importance of Agile Culture

Studies of social identity highlight the minimum conditions for in-group and out-group bias. Mere categorization into groups will instantiate a bias (Tajfel & Turner, 1986) which can further effect motivation to communicate and collaborate in organizational settings (Goodman & Olivera, 1998). Agile methodologies were seen to provide much more than mere categorization, however. The complex system of values, principles, and practices surrounding the term 'agile' comprises a rich culture for software development. Identification with the agile teams was thus seen to be especially

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

strong when compared to most non-agile teams, and commitment to collective goals and cohesiveness in the agile team environments was seen to be equally high.

While this could be due to the agile environment itself, identification by individuals in non-agile teams with agile culture and practices speaks to a certain 'stickiness' of the methodologies and culture towards members of the software development community. The value placed on simplicity and action, for example, was seen to resonate with people, and particularly with individuals dissatisfied with traditional software development efforts. The agile culture, by placing value on some things and not in others, was seen to mediate individual and group action in a way that supported successful interaction during software development, and to provide a common ethos about which individuals could unite. Thus 'agile' in this study was revealed to be more than a software development methodology, with significant aspects of cohesive teams in this study seen to stem from instantiation of an agile culture apart from the any particular practice.

The link between agile culture and cohesive agile teams suggests the importance of observing the interaction between agile culture and organizational culture. Participants in this study often discussed the difficulties faced when dealing with organizational culture or management styles that were inconsistent with agile values and culture. Seen in light of social identity theory, identification with one group culture is likely to interfere in identification with the other, thus reducing the benefits gained from a shared culture and in-group categorization.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Pitfalls of Self-regulation

Agile methodologies represent a structured form of self-regulating team well suited to the context of software development. In particular, the relative freedom of individuals to self-regulate within a structured plan was seen to heighten individual motivation and enjoyment, and to allow for high levels of innovation and opportunistic collaboration. Agile environments were seen in this study to support identification with the project team, commitment to a collective goal, and high levels of social awareness and accountability. Heightened awareness in the team environment, for example, was related to reduction in social loafing, or the tendency for individuals to expend less effort on a task when working in groups than when working alone (Latane et al. 1979).

A danger in such environments, however, is the dominance of 'concertive control' in the lives of individuals. In his article, *Tightening the Iron Cage: Concertive Control in Self Managing Teams (1993),* James Barker discusses the strong social forces present in post-bureaucratic structures such as self regulating teams. Barker explores a case study of an organization where normative (group) control in a self-regulating team becomes constrictive to the individuals within it, resulting in high levels of stress and disenchantment with collective efforts. Heightened social awareness, in this case, was seen to result in a heavy-handed system of social control, enhanced by the fact that team members themselves contributed to the development of team norms and rules. Thus what can be seen as the major benefit of self-regulating teams – their ability to motivate team members to work together in a flexible manner towards collective goals – can be waylaid by over-application of team controls.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Such application of team theory is unfortunate, given that initial work on self regulating teams conducted at the Tavistock Institute of Human Relations (Trist & Bamforth, 1951; Rice, 1958, 1963) was focused on individual well-being. Later teamworking initiatives, however, were seen to marginalize the democratic and 'Humanization of Work' (HoW) aspects of teamwork and focus instead on organizational performance measures (Mueller et al., 2000). Thus teamwork in current organizational contexts is often characterized by heightened organizational and system control over individuals akin to the authoritarian and bureaucratic control structures they were designed to replace (Barker, 1993; see also Edwards, 1981).

This small sample study of 22 participants has already revealed instances of burnout and stress in the agile environment, as well as of teams upholding norms un-supportive of flexible team functioning. Care should be taken, therefore, to critically appraise the use of agile methodologies, avoid misapplication of team-based controls, and take note of the interaction of agile methodologies with organizational culture and environment. The presence of a focus on productivity and performance to the exclusion of other considerations, for example, seems to have particular effects on the nature of self-regulating teams (see Cooney, 2004).

It's worth noting that agile culture was seen to be particularly valuable in preventing the development of self-regulating teams into controlling structures as opposed to flexible and purposeful structures. Conceptual tools such as 'do the simplest thing that will possibly work,' and 'you ain't gonna need it,' for example, can be seen to mediate normative effects and maintain team flexibility and efficiency. The action-based nature of agile culture can also be seen to reduce group effects leading to inertia in large

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

teams. It follows that implementation of the technical aspects of agile methodologies without concern for associated cultural aspects may leave room for the development of constrictive or unproductive team norms and activity.

An aspect of agile methodologies and culture that already seems to be underemphasized in the literature, for example, is the instantiation of agile teams to be reflective, adaptive, and self-regulating. Agile teams are often encouraged to change and improve based on team member input, and can be seen to be thus striving for optimal alignment of employees and technology as specified by socio-technical systems theorists (Emery & Trist, 1969). Cohesive agile teams in this study almost ubiquitously engaged in some level of team reflection or retrospection. By stepping back and evaluating their activity, such teams were seen to become aware of inefficiencies in their development efforts stemming from social or physical constraints, and to self-correct in a manner that supportive of optimal system performance.

Tools for Distributed Software Development

This study has highlighted the importance of socio-psychological activity and team member perceptions in the software development environment. Holistic awareness and feedback regarding the project, team, and product was shown to be particularly important to individual motivation and excitement, in supporting a collective culture of interaction, and in allowing for the team-wide involvement, interaction, and responsiveness associated with agile methodologies. This has implications for the design of tools for distributed software development, which seem primarily focused on detailed technical development and person-to-person interaction.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Exploration of group-based tools for software development seems a valuable focus for organizations seeking to support distributed agile teams. In particular, this study highlights the importance of tools that allow for the collection and display of normative information (see Whitworth, Gallupe, & McQueen, 2001), as well tools that display the linkages between individual efforts and collective efforts and goals. Tools providing awareness and feedback of whole team activity also seem essential to self-regulating and collective behavior. In particular, tools that provide frequent, engaging, and persuasive information display regarding limited areas of team-wide focus. Finally, this research suggests networked forms of communication, with interfaces to support informal, opportunistic, and context-rich information exchange and interaction.

One of the problems for distributed teams in light of this research is the relative difficulty of instantiating interpersonal relationships and rapport in computer-mediated environments. While such environments differ drastically from face-to-face environments, however, research has shown that the common conception of computer-mediated communication (CMC) mediums as inferior to face-to-face communication is not necessarily true. Social information processing (SIP) theory (Walther, 1992), for example, focuses on how individuals adapt to make use of the cue systems available in CMC to exchange interpersonal information. The SIP theory holds that communication of information over CMC is at first retarded when compared to face-to-face, but over time information accumulates, and interpersonal relationships can develop. SIP proposes different rates and patterns of impressions development over different media, where the process of forming impressions over CMC is not different from impressions forming in a face-to-face environment, but is usually slower.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Such research suggests that the adoption and use of computer-mediated communication mediums over time will increase the ability for individuals to communicate and relate to each other in the new environment. While distributed communication is slower than face-to-face communication, the major barrier in distributed software development teams seems to be the *absence* of communication, rather than the medium of communication. This research suggests that this may be due, in part to the absence of social forces present in the face-to-face environment are no longer supported by dispersed computer-mediated communication. A challenge, therefore, would be to explore means for the development, communication, and adoption of agile culture, norms, and ideals that are specifically suited to distributed software development and computer-mediated communication.

End Note

Exploration of agile software development teams from a socio-technical systems perspective revealed ways in which agile methodologies structure and mediate individual action and interaction in the context of software development. Such understanding has implications for the development and application of methodologies for team-based software design (which, it could be argued, can be held separate from the mechanics of software development). Notably, an understanding and awareness of social factors and controls in team-based software development seems valuable in supporting cohesive teamwork during software development, and in helping to avoid the pitfalls associated with group-based efforts.

Understanding of agile team activity was seen to involve complex phenomena crossing disciplinary boundaries; from software engineering, to social and organizational

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

psychology, to sociology. Research into such phenomena offered difficulties, ranging from a huge diversity of data, to a difficulty in placing findings within current academic literature. This study did, however, offer much insight into issues salient to today's software development practitioners, and into the application of psychological research to the practicalities of team-based software development. It is hoped that the results and discussion will provide a starting point for further inquiry and research regarding such issues.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

References Cited

Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2002). *Agile software development methods: Review and analysis*. Espoo, Finland: VT Publications.

Ackoff, R. (1999). *Ackoff's best, his classic writings on management*. New York, NY: John Wiley & Sons.

Ackoff, R., & Emery, F. (1972). *On Purposeful Systems*. Chicago, IL: Aldine Atherton.

Akgün, A. E., Byrne, J. C., Keskin, H., & Lynn, G. S. (2006). Transactive memory system in new product development teams. *IEEE Transactions on Engineering Management*, 53(1), 95-111.

Ambler, S. W. (2002). *Agile modeling*. New York, NY: John Wiley and Sons.

Ambler, S. (2003). *Agile database techniques: Effective strategies for the agile software developer*. Indianapolis, IN: Wiley Publishing Inc.

Ambler, S. (2005). Quality in an Agile World. *Software Quality Professional*, 7(4), 34-40.

Anderson, N. R., & West, M. A. (1994). The team climate inventory manual and users' guide. *Assessment Services for Employment*, Windsor, UK: NFER-Nelson.

Anderson, R. E., Carter, I. E., & Lowe, G. R. (1999). *Human behavior in the social environment: A social systems approach*. New York, NY: A. de Gruyter.

Arrow, H., McGrath, J. E., & Berdahl, J. L. (2000). *Small groups as complex systems: Formation, coordination, development and adaptation*. Thousand Oaks, CA: Sage.

ATIS Committee T1A1. (2001). *Definition: Information system*. Retrieved March 15th, 2006 from http://www.atis.org/tg2k/_information_system.html

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Auer, K., & Miller, R. (2001). *XP Applied* (The XP Series). Reading, MA: Addison Wesley.

Augustine, S., Payne, B., Sencindiver, F., & Woodcock, S. (2005). Agile project management: Steering from the edges. *Communications of the ACM*, 48(12), 85-89.

Badham, R., Clegg, C., & Wall, T. (2000). Socio-technical theory. In Karwowski, W. (Ed.), *Handbook of Ergonomics*. New York, NY: John Wiley.

Bandura, A. (1986). The explanatory and predictive scope of self-efficacy theory. *Journal of Social and Clinical Psychology*, 4(3), 359-373.

Bandura, A. (1989). Self-regulation of motivation and action through internal standards and goal systems. In Pervin, L. A. (ed.) *Goal concepts in personality and social psychology*. Hillsdale, NJ: Lawrence Erlbaum.

Bandura, A. (1991). Self-regulation of motivation and action through anticipatory and self-reactive mechanisms. In Dienstbier, R. (ed.) *Nebraska Symposium on Motivation 1990*, 38, 69-164. Lincoln, NB: University of Nebraska Press.

Bandura, A. (1992). On rectifying the comparative anatomy of perceived control: Comments on "cognates of personal control". *Applied and Preventative Psychology*, 1, 121-126.

Bandura, A., Cioffi, D., Taylor, C. B., Brouillard, M. E. (1988). Perceived self-efficacy in coping with cognitive stressors and opioid activation. *Journal of Personality and Social Psychology*, 55, 479-488.

Barker, J. R. (1993). Tightening the Iron Cage: Concertive control in self-managing teams. *Administrative Science Quarterly*, 38(3), 408-437.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Baron, R. A. & Byrne, D. (2000). *Social Psychology* (9th ed.). Needham Heights, MA: Allyn & Bacon.

Baskerville, R. L., & Stage, J. (1996). Controlling prototype development through risk analysis. *MIS Quarterly*, 20(4), 481-502.

Beck, K. (2003). *Test driven development: By example*. Reading, MA: Addison-Wesley.

Beck, K., with Andres, C. (2004). *Extreme programming explained: Embrace change* (2nd ed.) Reading, MA: Addison-Wesley Professional.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Martin 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 March 15th, 2006 from http://agilemanifesto.org/

Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M. (2005). Effects of Experience on Gaze Behaviour during Program Animation. *In Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop* (PPIG '05), Brighton, UK, p. 49-61.

Bellin, D. & Suchman Simone, S. (1997). *The CRC Card Book* (Object Technology Series). Reading, MA: Addison-Wesley Professional.

Bertalanffy, L. von. (1950). An outline of general system theory. *Britisch Journal of Philosophie of Science*, 1, 139-164.

Bertalanffy, L. von. (1968). *General system theory: Foundations, development, applications*. New York, NY: George Braziller.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bertelsen, O. W., & Bødker, S. (2003). Activity theory. In J. M. Carroll (Ed.), *HCI models, theories, and frameworks: Toward a multidisciplinary science.* San Fransisco, CA: Morgan Kaufmann.

Beyer, H. and Holtzblatt, K. (1998). *Contextual design: Defining customer-centered systems.* San Francisco, CA: Morgan Kaufmann.

Bion, W. R. (1961). *Experiences in groups.* London, UK: Tavistock Publications.

Boehm, B. (2002). Get ready for agile methods, with care. *IEEE Computer*, 35(1), 64-69.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed.* Reading, MA: Addison-Wesley.

Boylan, M. (2004). What have we learned from 15 years of supporting the development of innovative teaching technology? *Social Science Computer Review*, 22, 405-425.

Breer, P. E., & Locke, E. A. (1965). *Task experience: A source of attitudes.* Homewood, IL: Dorsey Press.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20, 10-19.

Bryant, B., Farhy, N., & Griffiths, A. (1994). *Self-Managing Teams & Changing Supervisory Roles.* Sydney: Centre For Corporate Change.

Bullock, J., Weinberg G. M. and Benesh M. (Eds.) (2001). *Roundtable on Project Management: A SHAPE forum dialogue.* New York, NY: Dorset House Publishing Company.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Campion, M. A., Medsker, G.J., & Higgs, A.C. (1993). Relations between work group characteristics and effectiveness: Implications for designing effective work groups. *Personnel Psychology*, 46, 823-850.

Card, S. K., Moran, T. P., & Newell, A. P. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Carroll, J. M. (1997). Human-computer interaction: Psychology as a science of design. *Annual Review of Psychology*, 48, 61-83.

Charbonnier, E., Huquet, P., Brauer, M., & Monteil, J. (1998). Social loafing and self beliefs: People's collective effort depends on the extent to which they distinguish themselves as better than others. *Social Behavior and Personality*, 26, 329-340.

Charmaz, K. (1990). "Discovering" chronic illness: Using grounded theory. *Social Science and Medicine*, 30, 1161-1172.

Charmaz, K. (1995). Grounded theory. In J. A. Smith, R. Harré & L. Van Langenhove (Eds.), *Rethinking methods in psychology*, p. 27-49. Thousand Oaks, CA: Sage Publications.

Chaston, I. (1998). Self-managed Teams: Assessing the Benefits for Small Service-sector Firms. *British Journal of Management*, 9(1), 1-12.

Chong, J., Plummer, R., Leifer, L., Klemmer, S. R., Eris, O., & Toye, G. (2005). Pair programming: When and why it works. *In Proceedings of the Psychology of Programming Interest Group Workshop*, Brighton, UK.

Coad, P., deLuca, J., & Lefebvre, E. (1999). *Java modeling in color with UML.* Upper Saddle River, NJ: Prentice Hall.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Coakes, E. (2000). Knowledge management : A sociotechnical perspective. *Proceedings of Operational Research in the 21st Century: Communications and Knowledge Management Conference (OR42)*, 14-28.

Cockburn, A. (2000). Characterizing people as non-linear, first order components in software development. In *Proceedings of the 4th International Multi-Conference on Systems, Cybernetics and Informatics (SCI2000),* Orlando, Florida.

Cockburn, A. (2002). *Agile software development.* Reading, MA: Addison-Wesley.

Cockburn, A. (2005). Crystal Clear: A Human-Powered Methodology for Small Teams. Reading, MA: Addison-Wesley.

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *IEEE Computer Magazine*, 34(11), 131-133.

Cooney, R. (2004). Empowered self-management and the design of work teams. *Personnel Review*, 33(6), 677-692.

Cordery, J. (2003). Team work. In Holman, D., Wall, T., Sparrow, P. & Howard, A. (Eds), The new workplace: A guide to the human impact of modern working practices, p. 95-114. Chichester, UK: John Wiley & Sons.

Coupland, D. (1996). *Microserfs.* New York, NY: Regan Books.

Creswell, J. (1997). *Research design: Qualitative and quantitative approaches.* Beverly Hills, CA: Sage Publications.

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004). Effective work practices for software engineering: Free/libre open source software development. In *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research* (WISER '04), Newport Beach, CA, 18-26.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Csikszentmihalyi, M. (2000). The contribution of flow to positive psychology: Scientific essays in honor of Martin E. P. Seligman. In J. E. Gillham, (Ed.), *The science of optimism and hope*. (pp. 387-395). Philadelphia, PA: Templeton Foundation Press.

Cummings, T. G. (1978). Self regulating work groups: A sociotechnical synthesis. *Academy of Management Review*, 3, 625-633.

Curtis, B. (1984). Fifteen years of psychology in software engineering: Individual differences and cognitive science. In *Proceedings of the 7th international Conference on Software Engineering* (ICSE 1984), 97-106. Piscataway, NJ: IEEE Press.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, 31(11), 1268-1287.

Damian, D., Chisan, J., Allen, P., & Corrie, B. Awareness meets requirements management: Awareness needs in global software development. In *Proceedings of the International Workshop on Global Software Development, International Conference on Software Engineering* (ICSE 2003), Portland, Oregon.

DeMarco, T., & Lister, T. (1999). *Peopleware: Productive projects and teams*. New York, NY: Dorset House Publishing Co., Inc.

Department of Defense. (1988). *Military standard: Defense system software development*. Washington, D.C: Department of Defense.

Department of Defense. (1994). *Military standard: Software development and documentation*. Washington, D.C: Department of Defense.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Department of Defense. (1995). *Standard for information technology software life cycle processes software development acquirer-supplier agreement.* Washington, D.C: Department of Defense.

Dourish, P., & Bly, S. (1992). Portholes: Supporting awareness in a distributed work group. In *Proceedings of the Conference on Human Factors in Computing Systems* (ACM CHI '92), Monterey, CA. 541-547.

Drucker, P. F. (1993). *Post-Capitalist Society.* New York, NY: Harper Business.

Dunphy, D., & Bryant, B. (1996). Team: Panaceas or prescriptions for improved performance? *Human Relations*, 49(5), 677-699.

Eby, L.T., & Dobbins, G. (1997). Collectivistic orientation in teams: An individual and group-level analysis. *Journal of Organizational Behaviour*, 18(3), 275-295

Edwards, R. C. (1981). The Social Relations of Production at the Point of Production. In Zey-Ferrel, M. & Aiken, M. (eds) *Complex organizations: Critical perspectives*, 156-182. Glenview, IL: Scott Foresman.

Emery, F. E. & Trist, E. L. (1969). Socio-technical systems. In Emery, F.E. (ed) *Systems Thinking*, 281-296. Harmondsworth, UK: Penguin.

Erickson, T., Kellogg, W.A. (2000). Social translucence: An approach to designing systems that support social processes. *ACM Transactions on Computer-Human Interaction*, 7, 59 – 83.

Evans, P., & Wolf, B. (2005). Collaboration Rules. *Harvard Business Review,* July-August, 96-104.

Festinger, L., Schachter, S., & Back, K. W. (1950). *Social Pressures in Informal Groups: A Study of Human Factors in Housing*. New York, NY: Harper.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Festinger, L. (1954). A theory of social comparison processes. *Human Relations*, 7, 117-40.

Fish, R. S., Kraut, R. E., Root, R. W., and Rice, R. E. 1993. Video as a technology for informal communication. *Communications of the ACM*, 36(1), 48-61.

Fitzgerald, B. (2000). Systems development methodologies: The problem of tenses. *Information Technology and People*, 13(3), 174-185.

Fowler, M. (2004). Is design dead? In G. Succi, & M. Marchesi (Eds.), *Extreme programming explained.* Boston, MA: Addison-Wesley.

Fowler, M. (2005). *The new methodology*. Retrieved January 29th, 2006 from http://www.martinfowler.com/articles/newMethodology.html

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). *Refactoring - improving the design of existing code.* Reading, MA: Addison-Wesley.

Fraser, S., Martin, A., Biddle, R., Hussman, D., Miller, G., Poppendieck, M., Rising, L., Striebeck, M. (2004). The role of the customer in software development: The XP customer - fad or fashion? *Proceedings of OOPSLA '04: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, 148-150.

Friedman, B., & Nissenbaum, H. (1996). Bias in computer systems. *ACM Transactions on Information Systems*, 14(3), 330-347.

Gharajedaghi, J. (1999). *Systems thinking: Managing chaos and complexity: A platform for designing business architecture*. Boston, MA: Butterworth-Heinemann.

Ghoshal, S. (2005). Bad management theories are destroying good management practices. *Academy of Management Learning and Education*, 4(1), 75-91.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Gibbs, W. W. (1994). Software's chronic crisis. *Scientific American*, 271(3), 72-81.

Glaser B. G. (1978). *Theoretical Sensitivity: Advances in the methodology of Grounded Theory*. Mill Valley, CA: Sociology Press.

Glaser, B. G. (1992). *Basics of grounded theory analysis: Emergence vs. forcing*. Mill Valley, CA: Sociology Press.

Glaser, B. G. & Holton, J. (2004). Remodeling Grounded Theory. *FQS, Forum Qualitative Social Research*, 5(2). Retrieved December 1, 2005 from http://www.qualitative-research.net/fqs-texte/2-04/2-04glaser-e.htm

Glaser, B. G. & Strauss, A. (1967). *The Discovery of Grounded Theory*. Chicago, IL: Aldine.

Glass, R. L. (2002). *Facts and fallacies of software engineering*. Reading, MA: Addison-Wesley Professional.

Goebel, C. J. (2002). *Extreme programming used to establish the culture of a high performance team: A management case study*. Menlo Institute, LLC. Retrieved November 18th, 2005 from http://www.menloinnovations.com/freestuff/whitepapers/highperformanceteams.htm

Goodman, P.S. & Associates (Eds.) (1982). *Change in organizations*. San Francisco, CA: Jossey Bass.

Goodman, P. S., & Olivera, F. (1998). Knowledge sharing via computer-assisted systems in international corporations. *Working Paper 98-17*. Pittsburgh, PA: Carnegie Bosch Institute.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Hackman, J.R. (1976). The design of self-managing work groups. *Technical Report No. 11*. New Haven, CT: School of Organization and Management, Yale University.

Hamilton, B. (2004). *NUnit pocket reference*. Sebastopol, CA: O'Reilly Media, Inc.

Hanks, B. (2003). Empirical Studies of Pair Programming. In *Proceedings of the 2nd International Workshop on Empirical Evaluation of Agile Processes* (EEAP 2003).

Hanks, B., McDowell, C., Draper, D., & Krnjajic, M. (2004). Program quality with pair programming in CS1. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (ITiCSE '04), 176-180.

Hanlan, M. (2004). *High Performance Teams: How To Make Them Work*. Westport, CT: Praeger Publishers.

Harrison, A., & Storey, J. (1996). New wave manufacturing strategies: Operational, organizational and human dimensions. *International Journal of Operations and Production Management*, 16(2), 63-76.

Hayes, S., & Andrews, M. (2000). An introduction to agile methods. Retrieved January 29th, 2006 from http://www.wrytradesman.com/articles/IntroToAgileMethods.pdf

Highsmith, J.A. III (1999). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, NY: Dorset House Publishing Company.

Highsmith, J. (2002). *Agile software development ecosystems*. Boston, MA: Addison-Wesley.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Highsmith, J. (2004). *Agile project management: Creating innovative products*. Reading, MA: Addison-Wesley Professional.

Hogg, M. A. & Vaughan, G. M. (2002). *Social Psychology* (3rd ed.) London, UK: Prentice Hall.

Hollan, J., Hutchins, E., & Kirsh, D. (2000). Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction*, 7(2), 174-196.

Huang, I.-C. (1998). Self-esteem, reaction to uncertainty, and physician practiced variation. *Social Behavior and Personality*, 26(2), 181-194.

Humphrey, W. S. (2001). Engineers Will Tolerate a Lot of Abuse. *IEEE Software*, 18(5), 13-15.

Husted, T. & Massol, V. (2003). *JUnit in action*. Greenwich, CT: Manning Publications.

Hutchins, E. (1995). *Cognition in the wild*. Cambridge, MA: MIT Press.

Ilgen, D. R., Hollenbeck, J. R., Johnson, M., & Jundt, D. (2005). Teams in Organizations: From input-process-output models to IMOI models. *Annual Review of Psychology*, 56, 517-543.

Jackson, P., Wall, T., Martin, R. and Davids, K. (1993). New measures of job control, cognitive demand, and production responsibility. *Journal of Applied Psychology*, 78(5), 753-62.

Jeffries, R. (2001). What is extreme programming? Retrieved March 15th, 2006 from http://www.xprogramming.com/xpmag/whatisxp.htm

Jeffries, R. (2004). *Passion*. Retrieved March 6th, 2006 from http://www.xprogramming.com/xpmag/jatPassion.htm

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Karau, S. J., & Williams, K.D. (1993). Social loafing: A meta-analytic review and theoretical integration. *Journal of Personality and Social Psychology*, 65, 681-706.

Katzenbach, J. & Smith, D. (1994). *The wisdom of teams: creating the high-performance organization*. New York, NY: Harper Business.

Kerth, N. L. (2001). *Project retrospectives: A handbook for team reviews* (2nd ed.). New York, NY: Dorset House Publishing Company.

Kidder, T. (1981). *The soul of a new machine*. Boston, MA: Back Bay Books.

Knights, D., & McCabe, D. (2000). Bewitched, bothered and bewildered: The meaning and experience of teamworking for employees in an automobile company. *Human Relations*, 53(11), 1481-1518.

Kobylinski R., Creighton, O., Dutoit, A. H., & Bruegge, B. (2002). Building awareness in global software engineering: Using issues as context. *In Proceedings of the 24th International Conference on Software Engineering* (ICSE '02), Orlando, Florida. 667-668.

Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM*, 38(3), 69-81.

Larman, C., & Basili, V. R. (2003). Iterative and incremental development: a brief history. *IEEE Computer*, 36(6), 47-56.

Latane, B., Williams, K., & Harkins, S. (1979). Many hands make light work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology*, 37, 822-832.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Lee, A. S., Liebenau, J., & DeGross, J. I. (eds.) (1997). *Information Systems and Qualitative Research.* London, UK: Chapman & Hall.

Leonard, A., with Beer, S. (1994). *The systems perspective: Methods and models for the future.* Retrieved March 6th, 2006 from http://www.futurovenezuela.org/_curso/6-sysmeth.pdf

Lessig, L. (1999). *Code and other laws of cyberspace.* New York, NY: Basic Books.

Levine, J. M., & Moreland, R. L. (1990). Progress in small group research. *Annual Review of Psychology*, 41, 585-634.

Lindvall, M., Basili, V. R., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., & Zelkowitz, M. (2002). Empirical findings in agile methods. In *Proceedings of Extreme Programming and Agile Methods* – XP/Agile Universe Conference 2002, Chicago, IL, 97-207. http://fc-md.umd.edu/fcmd/Papers/Lindvall_agile_universe_eworkshop.pdf

Marick, B. (2006). *Testing foundations: Consulting in software testing.* Retrieved April 15th, 2006 from http://www.testing.com/

Mark, G. (2002). Extreme collaboration. *Communications of the ACM*, 45(6), 89-93.

Martin, A., Biddle, R., & Noble, J. (2004). The XP customer role in practice: Three studies. *Proceedings of the Second Agile Development Conference*, 42-54.

Mayhew, D. J. (1999). *The usability engineering lifecycle.* San Mateo, CA: Morgan Kaufmann Publishers.

McConnell, S. (1999). *After the Gold Rush: Creating a True Profession of Software Engineering.* Redmond, WA: Microsoft Press.

McConnell, S. (2004). *Code Complete.* Redmond, WA: Microsoft Press.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. In *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon. 602-607.

McGrath, J. E., Arrow, H., & Berdahl, J. L. (2000). The study of groups: Past, present, and future. *Personality and Social Psychology Review*, 4, 95-105.

Merali, Y. (2004). Complexity and information systems. In J. Mingers, & Willcocks. L. (Eds.), *Social theory and philosophy of information systems,* p. 407-446. Sussex, UK: John Wiley & Sons Ltd.

Mierlo, H. van, Rutte, C. G., Kompier, M.A.J., & Doorewaard, A. C. M. (2005). Self-managing teamwork and psychological well-being: Review of a multilevel research domain. *Group & Organization Management*, 30(2), 211-235.

Miller L. A. (1974). Programming by non-programmers. *International Journal of Man-Machine Studies*, 6(2), 237-260.

Morhrman, S. A., Mohrman, A. M. J., & Cohen, S. G. (1995). Organizing knowledge work systems. In M. M. Beyerlein, D. A. Johnson & S. T. Beyerlein (eds.), *Advances in interdisciplinary studies of work teams vol. 2: Knowledge work in teams.* Greenwich, CT: JAI Press.

Mueller, F., Proctor, S., & Buchanan, D. (2000). Teamworking in its context(s): Antecedents, nature and dimensions. *Human Relations*, 53, 1387-1424.

Mugridge, R., Cunningham, W. (2005). *Fit for Developing Software: Framework for Integrated Tests* (Robert C. Martin Series). Upper Saddle River, NJ: Prentice Hall PTR.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Mumford, E. (1995). *Effective systems design and requirement analysis: The ETHICS approach.* Basingstoke, UK: Macmillan Press.

Muthusamy, S.K., Wheeler, J. & Simmons, B. (2005). Self-Managing Work Teams: Enhancing Organizational Innovativeness. *Organization Development Journal,* 23(3), 53-66.

Myers, M. D. & Young, L. W. (1997). Hidden Agendas, Power, and Managerial Assumptions in Information Systems Development: An Ethnographic Study. *Information, Technology & People,* (10) 3, 224-240.

Nakamura, J. & Csikszentmihalyi, M. (2003). The construction of meaning through vital engagement. In C. L. M. Keyes, & J. Haidt (eds.) Flourishing: Positive psychology and the life well-lived. Washington D.C.: American Psychological Association.

Nandhakumar, J., & Avison, D. E. (1999). The fiction of methodological development: A field study of information systems development. *Information Technology and People,* 12(2), 176-191.

Nardi, B. (1996). *Context and consciousness: Activity theory and human computer interaction. Cambridge.* MA: MIT Press.

Neck, C.P., and Manz, C.C. (1994). From groupthink to teamthink: Towards the creation of constructive thought patterns in self managing work teams. *Human Relations,* 47, 929-952.

Nicolescu, R., & Plummer, R. (2003). A pair programming experiment in a large computer course. *Romanian Journal of Information Science and Technology,* 6(1&2), 199-216.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Nicolini, D. (2002). In search of 'project chemistry.' *Construction Management & Economics*, 20(2), 167-177.

Nielsen, J. (1993). *Usability engineering*. Cambridge, MA: Academic Press.

Norman, D. A. (1988). *The design of everyday things*. New York, NY: Doubleday Press.

Norman, D. A., & Draper, S. W. (1986). *User centered system design: New perspectives on human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Nosek, J. T. (1998). The case for collaborative programming. *Communications of the ACM*, 41(3), 105-108.

Object Mentor, Inc. (2006). *JUnit.org*. Retrieved March 15th, 2006 from http://www.junit.org/index.htm

Opie, A. (2000). *Thinking Teams, Thinking Clients: Knowledge-Based Teamwork*. New York, NY: Columbia University Press.

Orlikowski, W., J. (1993). CASE tools as organizational change: Investigating incremental and radical changes in systems development. *MIS Quarterly*, 17(3), 309-340.

Paetau, M. 1996. Self-organization of social systems - a new challenge for organization sciences and systems design. *ACM SIGOIS Bulletin*, 17(1), 4-6.

Patton, J. (2002). Hitting the target: Adding interaction design to agile software development. *Proceedings of Object Oriented Programming Systems Languages and Applications Conference (OOPSLA '02)*. New York, NY: ACM Press.

Paulhus, D. (1983). Sphere-specific measures of perceived control. *Journal of Personality and Social Psychology*, 44, 1253-1265.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Peterson, C. M., & Seligman, M. E. P. (2003). Positive organizational studies: Lessons from positive psychology. In K. S. Cameron, J. E. Dutton & R. E. Quinn (Eds.), *Positive organizational scholarship: Foundations of a new discipline* (p. 14-27). San Francisco, CA: Berrett-Koehler.

Poppendieck, M. (2002). *Lean Thinking: The Theory Behind Agile Software Development.* Poppendieck.LLC. Retrieved November 18th, 2005 from http://whitepapers.zdnet.co.uk/0,39025942,60038859p,00.htm

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit for software development managers.* Reading, MA: Addison Wesley.

Pressman, R. S. (2005). *Software engineering: A practitioners approach* (6th ed.). New York, NY: McGraw-Hill.

Rayner, S. (1993). *Recreating the workplace: The pathway to high performance work systems.* Essex Junction: Oliver Wright Publications.

Reeves, J. W. (1992). What is software design? *C++ Journal*, Fall. Retrieved March 16th, 2006 from http://www.developerdotstar.com/mag/articles/reeves_design.html

Rennie, D. L., Phillips, J. R., & Quartaro, G. K. (1988). Grounded theory: A promising approach to conceptualization in psychology? *Canadian Psychology/Psychologie Canadienne*, 29(2), 139-150.

Rice, A. K. (1958). *Productivity and social organization.* London, UK: Tavistock Publications.

Rice, A. K. (1963). *The enterprise and its environment.* London, UK: Tavistock Publications.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Royce, W. (1970). Managing the development of large software systems. In *Proceedings of IEEE WESCON Conference*, 26, 1-9.

Rubin, H. A. (1993). Software process maturity: measuring its impact on productivity and quality. In *Proceedings of the 15th International Conference on Software Engineering*, p. 468-476. Los Alamitos, CA: IEEE Computer Society Press.

Russo, N.L. & Stolterman, E. (2000), Exploring the assumptions underlying information systems methodologies. *Information Technology & People*, 13(4), 313-327.

Salo, O., & Abrahamsson, P. (2004). Empirical evaluation of agile software development: The controlled case study approach. In *Proceedings of the 5th International Conference on Product Focused Software Process Improvement* (PROFES '04), Kansai Science City, Japan.

Sanna, L. J., & Parks, C. D. (1997). Group research trends in social and organizational psychology: Whatever happened to intragroup research? *Psychological Science*, 8, 261-267.

Sawyer, S. (2004). Software development teams. *Communications of the ACM*, 47(12), 95-99.

Scarbrough, H. & Kinnie, N. (2003). Barriers to the development of teamworking in UK firms. *Industrial Relations Journal*, 34, 2, 135-149.

Shneiderman, B. (1980). *Software Psychology: Human factors in computer and information systems.* Cambridge, MA: Winthrop.

Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM*, 20(6), 373-381.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Schwaber, K., & Beedle, M. (2002). *Agile software development with SCRUM*. Upper Saddle River, NJ: Prentice Hall.

Shore, J., with Woldrich, D. (2005). *Introduction to Fit.* Retrieved March 15th, 2006 from http://fit.c2.com/wiki.cgi?IntroductionToFit

Stapleton, J. (1997). *DSDM: The method in practice*. Boston, MA: Addison-Wesley Longman Publishing Company.

Stephens, M., Rosenberg, D. (2003). *Extreme Programming Refactored: The Case Against XP*. Berkeley, CA: Apress L.P.

Strauss, A. & Corbin, J. (1990). *Basics of Qualitative Research.* Newbury Park, CA: Sage

Sundstrom, E., DeMuese, K. P., & Futrell D. (1990). Work teams: Applications and effectiveness. *The American Psychologist*, 45(2), 120-133.

Sutcliffe, A. (2005). Applying small group theory to analysis and design of CSCW systems. In *Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering* (HSSE '05), St. Louis, Missouri, 1-6. ACM Press, New York, NY.

Tajfel, H., Billig, M., Bundy, R., & Flament, C. (1971). Social categorization and intergroup behaviour. *European Journal of Social Psychology*, 1, 149-178.

Tajfel, H., & Turner, J. C. (1979). An integrative theory of intergroup conflict. In W. G. Austin and S. Worchel (Eds.), *The Social Psychology of intergroup relations.* Monterey, CA: Brooks/Cole.

Tajfel, H., & Turner, J. C. (1986). The social identity theory of inter-group behavior. In Worchel, S. & Austin, L. W. (eds.). *Psychology of Intergroup Relations.* Chicago, IL: Nelson-Hall

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Taylor, F.W. (1947). *Shop Management and Principles of Scientific Management.* New York, NY: Harper & Row.

Teasley, S. D., Covi, L. A., Krishnan, M. S., & Olson, J. S. (2002). Rapid software development through team collocation. *IEEE Transactions on Software Engineering*, 28(7), 671-683.

The Standish Group. (1994). *Charting the Seas of Information Technology.* Dennis, MA: The Standish Group.

The Standish Group. (2004). *The CHAOS Chronicles Version 3.0.* West Yarmouth, MA: The Standish Group.

Thorn, B. K., & Connolly, T. (1987). Discretionary data bases: A theory and some experimental findings. *Communication Research,* 14(5), 512-528.

Trist, E. L. & Bamforth, K. (1951). Some social and psychological consequences of the longwall method of coal getting. *Human Relations*, 4(1), 3-38.

Trist, E. L., & Joyce, W. F. (1981). The sociotechnical perspective: The evolution of sociotechnical systems. In A. H. Van de Ven (Ed.), *Perspectives on organization design and behavior*, p. 19-75. New York, NY: Wiley and Sons.

Turner, J. C., Hogg, M. A., Oakes, P. J., Reicher, S. D. & Wetherell, M. S. (1987). *Rediscovering the social group. A self-categorization theory.* Oxford, UK: Blackwell.

Turner R., & Boehm B. (2003). People factors in software management: Lessons from comparing agile and plan-driven methods. *Crosstalk*, 4-8.

Two, M.C, Poole, C., Cansdale, J. & Feldman, G. (2005). *NUnit.* Retrieved March 15th, 2006 from http://www.nunit.org

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Van der Veer, G. C., Lenting. B. F., Bergevoet, B. A. J. (1996) GTA: Groupware task analysis – modeling complexity. *Acta Psychologica*, 91, 297-322.

Vicente, K. J. (1999). *Cognitive work analysis: Toward safe, productive, and healthy computer-based work.* Mahwah, NJ: Lawrence Erbaum Associates.

Vogler, C. (2000). Social identity and emotion: the meeting of psychoanalysis and sociology. *The Sociological Review*, 48(1), 19-42.

Warr, A., & O'Neill, E. (2005). Understanding design as a social creative process. In *Proceedings of the 5th Conference on Creativity & Cognition*(C&C '05), London, UK. 118-127.

Wastell, D. G. (1999). Learning dysfunctions in information systems development: Overcoming the social defenses with transitional objects. *MIS Quarterly,* 23(4), 581-600.

Wegner, D. M. (1986). Transactive memory: A contemporary analysis of the group mind. In Mullen, B. & Goethals, G.R. (Eds.), *Theories of Group Behavior*, p. 185-205. New York, NY: Springer-Verlag.

Weinberg, G. M. (1971). *The psychology of computer programming.* New York, NY: Van Nostrand Reinhold Company.

Wenger, E., McDermott, R., & Snyder, W. M. (2002). *Cultivating communities of practice.* Boston, MA: Harvard Business School Press.

Williams, K. D., & Karau, S. J. (1991). Social loafing and social compensation: The effects of expectations of coworker performance. *Journal of Personality and Social Psychology*, 61, 570-581.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Winner, L. (1986). *The whale and the reactor: A search for limits in an age of high technology*. Chicago, IL: University of Chicago Press.

Whittaker, S., Frohlich, D., & Daly-Jones, O. (1994). Informal workplace communication: What is it like and how might we support it? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating* Interdependence (CHI '94), Boston, MA.

Whitworth, B., Gallupe, R. B. & McQueen R. (2001). Generating agreement in computer-mediated groups. *Small Group Research*, 32(5), 625-665. http://brianwhitworth.com/sgr01.rtf

Womack, J. P., Jones, D. T., & Roos, D. (1990). *The machine that changed the world: The story of lean production*. New York, NY: Rawson Associates.

Yourdon, E. (2003). *Death march* (2nd ed.). Upper Saddle River, NJ: Prentice Hall PTR.

Zelkowitz, M., Yeh, R., Hamlet, R., Gannon, J., & Basili, V. (1984). Software engineering practices in the U.S. and Japan. *IEEE Computer*, 17(6), 57-66.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendices

Appendix A: Ethics Proposal

Purpose of Study

The proposed research seeks to investigate the subjective experience of individuals (practitioners) in agile software development teams. Grounded theory was chosen due to the understudied research area and the nature of software development, which is suited to qualitative exploration. The study will be conducted through semi-structured one-on-one interviews, with topics involving software development practices employed at the individual's place of work, and the effect of these practices on the experiences of the individual and their relationships with other team members. The purpose of the interviews is to gain conceptual understanding of actions and interactions occurring in software development teams. Particular focus will be put on agile software development methods, and any emergent phenomena that may result from the use of such methods. Findings will be written up in a qualitative report. This proposal concerns the protocol regarding the practitioner interviews.

Method

Participants. It is expected that approximately 15-20 participants will be involved in this study. The exact number will depend on the conceptual requirements found during the grounded theory process.

Materials. The materials include: an informed consent form (See Appendix B), an unstructured interview guide (See Appendix C), a debriefing form (See Appendix D), and a voice recorder.

Procedure. Participants will be recruited through interaction with software development and agile communities. This will include face-to-face recruitment at events such as conferences and workshops, notices of recruitment (see Appendix E) sent directly by email or through relevant mailing lists, and through snowball sampling with existing participants in the study.

Participation will be voluntary, and secured on an individual basis. The cost to participants will be minimal, simply involving time. Participants will be introduced to the study and asked if they would be willing to take 1 hour to talk about their experiences on an agile team. Once recruited, a time and place convenient to the participant will be arranged, for example in a meeting room at their place of work. Care will be taken to find a room or space that is private enough where they can speak freely, and without worrying that others can overhear.

Before the interview, participants will be asked to read and sign the informed consent form (see Appendix B), and the interview will commence using the interview guide (see Appendix C). Interviews will be recorded given the consent of the participant.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Tools and artifacts used in software development, such as diagrams, charts, posters, software development environments may be explored to support participant explanations. Understanding of items related to my area of study will be gathered during interviews, at which time additional understanding may be gained by examination of the artifacts available at the interview location. The contents of tools and artifacts analyzed will be kept completely confidential and only findings will be reported. For example, if a participant describes the content of a wall chart as relevant to their experience, the report could read: 'the participant makes use of a wall chart containing a prioritized and color coded list of user stories.' If necessary nondisclosure agreements will be signed and abided by.

After the interview participants will be given a debriefing form explaining the way their information will be used (see Appendix D). Participants will be asked if they would like to be involved in follow up activities, such as additional interviews, or review of the research findings. Those participants who wish to be involved in the review of research findings will be sent a manuscript on which they can comment and send back.

Summaries of research findings will be made available to participants at least once, and perhaps several times during the course of the study, along with a request that interested participants validate findings and provide any additional feedback.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix B: Informed Consent Form

Masters Research into communication and collaboration in agile software development teams

Thanks you for agreeing to be interviewed for this research. Prior to conducting the interview, Carleton University requires that I obtain your written informed consent. This consent is a normal part of any research project.

Research Personnel

Elizabeth Whitworth
Principle Investigator
Carleton University
(613) 520 2600 Ext. 6628
elizabethwhitworth@gmail.com

Robert Biddle
Faculty Sponsor
Carleton University
(613) 520 2600 Ext. 2026
robert_biddle@carleton.ca

Nature and purpose of research

This research project contributes towards the requirements for a master's degree. The purpose of this research is to investigate the subjective experience of working on a software development team, with particular focus on agile communication and collaboration. The main source of data for this study will be interview data, but management and communication mechanisms such as diagrams and charts are also of interest, along with tools or environments used to develop software.

Duration and Locale

The interview will take approximately 1 hour, and will occur in a location convenient to you. Please don't hesitate to let me know of any reservations you have regarding comfort or privacy during the interview.

Potential Risk/Discomfort

There is no psychological or physical risk associated with this research.

Audio Taping

Interviews will be audio recorded given your consent. Recordings will be reviewed by research personal for the purpose of this study only. Recordings will be destroyed once transcribed or upon completion of this study.

Anonymity/Confidentiality

All raw data will be kept confidential to myself and my supervisor Robert Biddle. Written interview notes, interview recordings and transcriptions will be stored at my home and/or office at university, and will be destroyed on completion of the project. The final research report will be published as a master's thesis held at the Carleton University Library. Conferences and journal publications may also be written based on this research.

Research reports will not identify you or your employer. You may request to review any written notes that result from the interview to ensure factual material is recorded correctly, and there will be an opportunity to review the final research report to validate findings and interpretation of material.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Right to Withdraw

Please understand that you have the right to withdraw from the interview at any time, without any explanation as to the reason for withdrawing, and the right to refrain from answering any question you do not want to. You are also free to completely withdraw from this research project without penalty or explanation up to the 1st of August, 2006.

Agreement and Signatures

I have read and understand the above terms and I understand the conditions of my participation. I have had the opportunity to ask questions and have had them answered to my satisfaction.

My signature indicates that I agree to participate in this research project, and consent to the collection and use of my perceptions, experiences, opinions and information in this research.

Do you agree to have interviews audio recorded? Yes / no

Participant's Name: _____

Participant's Signature: _____

Date: _____

Researcher's Name: _____

Researcher's Signature: _____

Date: _____

If you require any further information please feel free to email me at elizabethwhitworth@gmail.com. Alternately you may contact my supervisor Robert Biddle (robert_biddle@carleton.ca, 613-520-2600 Ext. 2026).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix C: Sample Interview Questions

Experience

Years of experience in an agile software development team: ___
Years of experience with software development teams: ___
Roles held: _____

What is your general level of excitement regarding agile software development?

Not Very Excited				Very Excited
1	2	3	4	5

Think of a time when you or your team was particularly motivated or excited about a project or part of a project. Describe this to me...

Describe your team environment.

What do you like about your team environment?
What do you dislike about your team environment?

Are you aware of what the rest of the team is working on? If so, how are you aware of this? Do you think this awareness/lack of awareness affects your work?

What agile practices do you feel are the most useful in your software development efforts?

For each practice explore in greater detail...

- What is the purpose of this practice?
- How does this practice help you develop software?
- How does this practice help you communicate with other team members?

Which agile practices do you feel are the least useful?

For each practice explore in greater detail...

- What is the purpose of this practice?
- How and why does this practice fail to be of use?

How have your team interactions been effected by the use of agile?

- How do you feel about the other members of the team? Why do you feel that way?
- How do you feel about the project in general? Why do you feel that way?
- How do you feel about the work you do? Why do you feel that way?
- How do you feel about your working environment? Why do you feel that way?

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix D: Debriefing Form

The research I am conducting is aimed at gaining and understanding of the subjective, or 'lived' experience of working in an agile software development team. The study is designed to elicit information that could better inform the application of agile in non-traditional contexts, such as in large or distributed teams, or on projects with strong user-centered design components.

The project is scheduled to be completed by September, 2006. Unless otherwise requested, I will send you a summary of my research findings and interpretations at the beginning of August. At this point you will get the chance to validate your contribution to the research and offer additional feedback.

Please let me know if you are interested in receiving earlier summaries of my research regarding your interview in particular and/or the research project as a whole, and I will do my best to accommodate.

Thank you for participating in my master's project! Your time and effort are greatly appreciated.

If you have any further questions regarding this research, please contact:

Elizabeth Whitworth
Principle Investigator
Carleton University
(613) 520 2600 Ext. 6628
elizabethwhitworth@gmail.com

Robert Biddle
Faculty Sponsor
Carleton University
(613) 520 2600 Ext. 2026
robert_biddle@carleton.ca

If you have concerns about the ethics of this research, please contact the principle investigator above, or:

Dr. Mary Gick
Chair, Department of Psychology
Carleton University
(613) 520 2600 ext 2664
Mary_Gick@carleton.ca

Dr. Janet Mantler
Chair Ethics Committee for
Psychological Research
Carleton University
(613) 520 2600 ext 4731
Janet Mantler@carleton.ca

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix E: Announcement for Recruitment

Participants will be primarily recruited through face-to-face requests during participation in agile community events such as conferences and workshops. The research project will be briefly described, and prospective participants will be asked if they would be willing to participate in interviews.

If necessary, notices of recruitment will be posted in areas frequented by practitioners in the software development community. These will most likely be posted to electronic forums and mailing lists.

All notices of recruitment will contain the following information:

Do agile practices give you a warm fuzzy feeling?

Well then I would like to talk to you.

My name is Elizabeth Whitworth. I'm a master's student in human-computer interaction, and I am curious to study the subjective experience of working in software development teams. I'm particularly interested in practices that may contribute to high performance teams.

While I am focusing on agile practices, I am happy to talk to developers in non-agile teams. Basically I'm looking to hear interesting stories about team communication, collaboration, and motivation during any software development project.

If you would like to participate in this study, please send an email to elizabethwhitworth@gmail.com and I will contact you to arrange a short (and perhaps even enjoyable!) interview.

I hope that the findings of this study will help us better create team 'synergy' in non-traditional contexts, such as large or distributed teams.

Thank you for your interest, and I hope we can be in touch!

Warm regards,

Elizabeth Whitworth
elizabethwhitworth@gmail.com
hot.carelton.ca/~ewhitworth

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix F: Analysis Of Results

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Introduction

The focus of this research was motivation and excitement in software development teams. A distinction is made in this study, therefore, between 'cohesive' agile teams characterized by strong 'team feeling' and excitement, as opposed to non-cohesive software development teams. Reference to cohesive teams throughout the rest of this document denotes teams identified by participants as well functioning teams that 'clicked,' 'gelled,' or 'truly worked together' to successfully develop software. Impressions gathered from interview participants suggest that such teams were also more productive and 'high-performing.' My research interest, however, revolves around the *experience* of the individual rather than any external measure of success such as team productivity. It should be noted that the view of agile in this study is based on, but not limited to, extreme programming practices, and is focused on agile teams that are as close to 'ideal' as possible.

Whole Team Awareness, Feedback, and Focus

Common Knowledge – a basis for collective action

Certainty in the Agile Team Environment

Members of cohesive agile teams were much more likely to use strong language such as 'will' and 'know' when describing what was occurring in their team environment. This could be compared to members of non-agile or non-cohesive teams, who were more likely to use vague language such as 'probably' or 'feel.' This was seen to reflect the presence or absence of discipline and rigor in the team environment.

> *Unit tests were being written, the nightly build was important...actually we had several builds a day. And that was clear to everybody. And there was no discussion necessary on that point. And the discipline to the internal people came fast enough. Because they listened and paired, and they saw that it works. And we had strong people in every team. Just pushing it.*
>
> (X.4.29)

> *The standards' higher. In terms of how the developers hold each other accountable for you know somebody check something and it doesn't have adequate tests; then they will hear about it, from more and more people.*
>
> (T.1.31)

In the first case, for example, developers were strongly encouraged to adhere to agile standards such as unit testing and integration of individual code into the collective code base (by running a 'build'). Such team discipline was related to feelings of certainty regarding whole team activity in the agile environment:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

It's invigorating, because for the first time in your career you know exactly what is expected of you

(O.2.15)

Another factor in development of team certainty was flexibility of the agile plan, which could be realigned to suit changing needs on the project and re-communicated to all members of the team:

The way that we are working now there are very few if any crisis. That was not the case with other projects I have been part of where we didn't use this type of process. There could easily be a crisis that...couldn't get the features in; you ran out of time; there was something is really buggy. There was a lot more opportunity for crisis and crisis management in other ways of working. With this because it's so...the cycles are so short and there is so much revisiting of the plan it really removes a lot of the ability for a crisis to appear because you know right away that something is going to take a whole lot longer or something is not working. And if something is not working you are going to know within the next week or two and you can be scheduled and you can refit everything accordingly.

(T.3.32)

Adherence to the specified plan in the team environment was seen to increase security on both an individual level, and on a team level. It allows agile teams to avoid situations where one party is waiting for another, which not only slows down the software development process, but seems to increase tension, annoyance, and resentment amongst team members.

I don't think that we would have been so consistent in our releases if we had done it otherwise ... The main point is that I feel more secure about what I'm doing; very strongly. It's also what they tell you the basic reasons behind unit testing, continuous integration. And it's basically that. And I really feel it like that. You really prove after two weeks that this thing is really working, in a more or less stable way.

(X.4.11)

Certainty was also seen to stem partially from the fact that increased communication in the agile team environment constantly reaffirmed team focus and increased awareness of others:

And then as time goes on people, you know, they don't need to, they almost don't need to vocalize it because everybody knows by that time because you've gone around the room and etc [they know] that you're really interested in that.

(L.4.39)

Thus practices such as daily stand-up meetings, where each team member would be required to give an update of their current progress, tasks, and any problems that they were having, would reiterate team knowledge of both the project, and of each member of the team.

Certainty in the team environment was also related to the regular iterative delivery of working software, which in an agile environment generally occurs weekly, bi-weekly, or

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

monthly. Such repeated delivery of software allows team members to establish a history of interactions on which to base trust in both individual ability and the ability of the team to work together. In discussing a crisis situation in his team, one participant notes:

> *No. We will do it. We always do it.*
>
> (O.2.13)

This could be held in comparison to software development environments where working software is delivered only at the end of a long development period such as six months, a year, or longer.

Agile practices also involve team members that are dedicated to the team. Part time team members were also associated with loss of trust, in addition to a lack of 'team feeling':

> *It's like when I was saying when [a team member] wasn't on the [project full time] and they wanted his help and he is not there and then they feel betrayed kind of because he is not there and they need him...If they are not dependably there, you don't get that close feeling, emotion of being able to depend on someone. So that's kind of what's been coming out of when I have been asking people that.*
>
> (T.4.69)

Whole Team Awareness

Awareness and feedback surrounding the team *as a whole*, as opposed to individual or partial team communications, was seen to allow seemed a sense of 'common knowledge' in the team environment. Development of this common knowledge was essential for the functioning of software development teams as cohesive units, as it could then be used as a basis for action that has been 'approved' by the whole team:

> *And so at the next retrospective you will say well things did happen. You know, we did change things ... it could even be changing you know the time or the type of the kickoff meeting or the standup is too long or you know something. And its basically having the sort of collective thing of well, agreement that there is a problem and then agreement that there is just a way we can improve things rather than having a sort of whinge and then you know not being able to do...or feeling powerless you know in terms of - oh we have to live in this environment you know.*
>
> (L.4.13)

> *In that project we did a lot of pairing. And a lot of pairing of business people with technical people. And that worked quite well...that worked very well because there was a lot of respect for everyone. It was clear who knew what better. And this was the way it was and it was okay for everybody*
>
> (X.4.13)

Collective awareness of activity in the team environment seemed to give participants an increased ability to engage in initiatives that affected the whole team:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

It's easier for me to ask some of the bigger questions about the product. Then I don't know what other forum we would have. I mean I could talk to the director directly and say 'Well what's happening with this particular contract or what's happening with? We were talking to a possible customer and what happened with that?' And he could tell me directly what happened but then no one else would know. Do you know what I mean? I did mention something about to possibly working in the idea of customer fieldtrips for us to go to customer sites and you know I could bring that up with him directly but it's better to bring it up in a general forum I think because then everybody will see what's going on

(T.2.30)

Self-Regulating Team Behavior

Whole team awareness also seemed important in allowing self-regulating team behavior. In the following excerpt, for example, the project leader uses an 'information radiator,' or highly visible shared team artifact, to ensure that developers were completing team development tasks as opposed to moving on to the next new thing:

...Not starting anything new but finishing the stuff that they had. And by having it up here on the board, then everyone knew that it was sort of fair, you know. They knew that things needed finishing, and they knew that, you know, that the work had to be allocated to someone. So someone getting lost by...I mean it's very typical of developers, but no-one really wants to do all the finishing, you know, tidying up stuff. They all want to do the new stuff.

So it's just bringing a little bit of, ah, sort of visibility for the things that needed finishing for the stories to get done. And in a way that was sort of fair to everyone. You know I wasn't sort of picking on anyone. It is up to everyone to work out what they were going to do next, but they just understood that...someone in the team...had to take those nasty – chasing up someone to find out what the answer to some question was, to get something done – tasks.

(L.5.28)

The reasons for increasing visibility of team goals and tasks included ensuring that team members knew distribution of work was fair, and that this work had to be done by *someone* in the team.

Study data supports the fact that team members who have bought into the team goal are more likely to happily pick up on tasks that are less personally enjoyable if they know that other team members were occupied with other tasks:

And you found they would do that, just pick it up and do it?

Yeah because you know ask people...you know about one person who would be stuck on the task and there would be other people who would be just looking for other stuff, they had just finished a task and looking for a new task to do. And just saying well, you know, 'The next thing to do is this. So who out of the people who are available is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

going to do it?' I think there were even, you know, even more the sort of people who wanted to do new stuff than normal. I think that was a part of it.

(L.5.28)

Thus increased visibility of work to be done in the agile team environment was seen to reduce the occurrence of social loafing, where undesirable tasks are 'lost' and left for others to do:

...And I think another thing was just the visibility. Because they hadn't been tracking in a sort of a transparent way, they were getting to the end of iteration and then realizing that, oh yeah we haven't done that bit. Whereas if they had been able to see it all the way through, then maybe they would have, um, picked up on it themselves. It's just because they sort of collectively forgot that this had to be done to get that story finished, you know, they would all be thinking that someone else would do it, and it wasn't entirely clear to them who was doing what and what needed to be finished.

(L.5.28)

Team members discussed in this study were most likely to take responsibility of tasks when they knew what needed to be done, knew the capabilities of others in the team, and were aware what other people are doing. If it became clear that no-one else was available or more qualified, then they would pick up even the most tedious task and do it quite willingly. Individual responsibility was therefore seen to be highly dependent on awareness of the entire team. Lack of awareness of the whole team increased the chances that tasks or issues would fall through the gaps and not get completed or resolved - not necessarily because no-one was aware of them, but because of lack of awareness that no-one else was available to take responsibility for the task.

Team Awareness of Individual Actions

The knowledge that other team members were present and noticing individual actions in the team environment was seen to have a strong mediating effect on the actions of individuals in the team environment. The strength of team meetings as a forum for collective action in agile teams was especially surprising. The value of such meetings seemed related to daily affirmation of team goals and commitment from all the members of the team, and in receiving knowledge to support the fact that everyone in the team was doing their part towards the common goal rather than working towards personal goals:

And I think not having that daily meeting where you kind of affirm that everybody is on the same team and you have to tell people what you are doing for the good of project like you can't sit there...if you have a daily meeting you can't sit there and say, 'Well, actually I have just been working on my own thing.' <laughs> Or it's a little harder to do that right, like 'Well I haven't been working at project at all!' <laughs>

(T.4.39)

Participants in cohesive teams often mentioned the importance of the activity of checking up on team progress on a daily or weekly basis. Agile practices seem to structure and motivate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

individual behavior by providing regular forums in which the entire team is aware of what they have been doing.

Part of what allows for regular checking on progress in the agile environment is the limitation of expectations into plausible, achievable, and noticeable chunks. Agile teams would generally assign relatively limited responsibilities to individuals, once they were agreed upon by the entire team. After a short period, when things are still fresh in people minds, the team would revisit what had been done towards these goals:

> *And that's where the positive thing comes out of it because what happens is you might decide to prioritize the ownership of these actions that you are going to take to further mitigate or improve the bad situations. And so somebody has got to have responsibility for maybe just 2 out of 10 of them you know just however many you know of the...And what happens is that the next stretch is like somebody will go over and say well this, this and this was dealt with is everyone is everybody satisfied that it was dealt with it.*
>
> (L.4.11)

Team members have little excuse not to have completed tasks, and express a strong aversion to showing up to team meetings without showing progress towards the assigned goal:

> ***Why is there this higher priority on making the team look good rather than making yourself look good?***
>
> *Respondent: hmmm...one thing...hmmm. Because it is rewarding to success as a team. To share the success with team members. And on the other side we have reflection meetings. And so what we are doing is looking at the team goal, and deciding as a team that we reached them. And if we didn't meet a team goal, then talking about why we didn't reach it. So if I didn't contribute to reaching a goal, then it becomes very visible. And so...I am not very much into rewards or praise. I think, in this case it's important that everyone knows that you don't get praise for doing your things well, but you get praise for making the team succeed. And it simply doesn't feel good, at the reflection meeting, to say...ok I didn't take care of the team goal, I did this task. It wasn't that important to the goal, but it was important to me...so yeah*
>
> (X.3.44)

In this way, much of the freedom associated with the agile team environment is balanced by the presence of collective team awareness and purpose. Repercussions for bad quality or incomplete work in the agile team environment tend to be social rather than punitive. These can come from other members of the team questioning or making fun of actions, or may be present only in expectations of disappointment or disapproval from the team as a whole.

Team awareness and praise for individual interactions was seen to provide a very strong motivator:

> I told you about the button that we have to press? Well in fact we also have a website that you can go to and press the button there and get the same impact. And I still

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

like, I think to get up and press the physical button. Because one thing is that it is just more fun. Cool in some kind of way. And the team notices who is pressing the button. And even though when you press the button on the website there is some sound played, and the state of the traffic light changes. It's often something that the rest of the team won't notice as much. But when I am going there to press the button, they just look at me, and they notice the kind of team directions that's missing with the software tools...the interaction with the team, is something that is very important. I feel as part of the team. And that's happening while I walk up.

(X.3.6)

Audience

This study also highlighted the effects related to having an audience for team communications in an agile environment. Tangible excitement from other team members, for example, and the knowledge that others saw value in the individual contribution to the team goal was seen to be a strong motivator:

I know they really like being there because they get to hear first hand what the requirement is and they can start thinking right away about, you know, how I can implement that and so forth and I know they do that as well because there are times when we have to sort of put the brakes on implementation discussion within the requirements meeting because they just, you know, they're hearing it, they're getting excited, and so that happens quite a bit. But I know they like being involved.

(T.5.12)

Awareness of the audience of a communication has the additional effect of changing the nature of team communication. When team members are aware, not only of their own responsibilities, but of the responsibilities of others and the current context of interaction, there they are much more likely to respect others when delivering team communications. For example, in the following example, where team communication is kept to a minimum to make efficient use of team resources of time and attention:

I know that for our group we tried very hard to make sure that what we were saying would be interesting for everybody. Like we didn't bog down things with things that we cared but nobody else cared about and I think...except for a couple of developers <laughs>, most of the people would do that too. They wouldn't say, you know, 'I have just answered an Email' or something like that. They would just say those few things that were really kind of generally interesting. And so that kept that momentum going.

(T.4.4)

The fact that all action was occurring in a team context seemed to have other effects. Further study would be required to establish exactly what these were, but the following participant provides an example:

I think the open concept area actually helps to lower the noise.

Really?

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *I find that in my experience when you have big cubicles where you can't see anybody, it's, they are out of sight, they are out of mind. So, volume increases if you can't see anybody. But if you can see all the people around you, you can tell if you are being loud because, you know, they will twitch their ear because you just, you know, were talking too loud, then it's an indication that those people can go into a meeting room that is right beside our area. That was surprising to me, everyone, and myself included, thought that you get into a big open space like that, that it would be really loud. But I think it just, you have more respect for the people around you. You are aware of the people so you know to restrain yourself.*
>
> (T.3.2)

Thus social awareness seems important in allowing team members to regulate their behavior based on others. Awareness and feedback from others was seen to be essential in *allowing* team members to be respectful of others; in an environment where individuals are less aware of their mean members there is less opportunity to show respect in this manner.

Distributed Knowledge and Leadership

Agile culture and practices encourage the distribution of both knowledge and responsibility across the entire team. Agile teams focus on reducing team and organizational dependency on singular experts regarding a particular skill or aspect of the current software development effort. Practices most supportive of this are pairing, where two team members work together on the same task, and work in a shared team room, where all team members sit together in the same room with no dividing walls between them. This study highlighted a number of positive effects related to such practice.

The ability of a team to absorb the loss of a team member was seen to increase the sense of control regarding their ability to deal with unforeseen circumstances, as well as to decrease the feeling of pressure or responsibility on the part of the individual who would otherwise be the gate-keeper of essential project information:

> *You know I work on a reduced work-week. I actually don't work Fridays. I work 4 days a week. So it's always nice for me to have someone who can, who vaguely knows what's going, what I have been working on for the Friday; if something breaks or goes probably wrong and there is always somebody else to fill in or who knows roughly what I have been doing. So I find that very useful for me.*
>
> (T.2.9)

This helps avoid a number of situations, such as: when one person is 'stuck' in a particular role against their wishes because there is no-one else who can fulfill it; where one person creates a bottleneck, increasing tension between team members; or where one person is overly bothered by others who require access to information only they can provide. For example:

> *Part of the team culture would be the lead-engineer...people know to go to him. As the person who knows who else knows. So he's been working there for about 10 years.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

And he knows what everybody else knows. It frustrates him that he's the one that everybody goes to...but most operations have people like that

(L.1.13)

The fact that knowledge is more evenly distributed throughout the team, combined with a shared knowledge of who knows what in the team environment, seems to decrease the likelihood that one particular 'leader' will emerge as the focal point for team activity and decision-making. It is more likely, in this case, that leadership will shift from one person to another depending on the particular task requirements, and that there will be more of a feeling of 'equal partnership' in the team environment where each team member can contribute to whatever part of the project they would like to be involved in:

And so we don't have any formal leaders, but I think we have leaders emerge at specific situations... And we have one developer who is very strong on user interface design, so whenever it is about user interface design then he will take lead. And...but it's very informal and yeah I think the team wouldn't feel comfortable with true...with a formal leader, with a designated leader. Because you just have so much, um, there is so much talent in the team that it wouldn't be appropriate to talk about a single leader.

(X.3.59)

Group Action and Inaction

A major problem in the software development team environment is generating group agreement and, perhaps more importantly, *action.* While the phenomena surrounding this topic are not fully understood, agile practices were seen to greatly increase the chances of group agreement and action. Many study participants mentioned situations were no action was taken despite group agreement:

And I used to be at this small consulting company that was about 20 people. And it was too... they would try to do everything like half democratic. And it was actually the guys that I did this project – this agile project. So agile. They were extremely good people. But the company itself didn't actually work. Because it was 'Yeah we should do that, yeah we should. Yeah ok. Let's talk about it at the next annual meeting because there is something we could do." And at the next annual meeting – 'Yeah we should do this. Yeah. Well maybe you can make a plan.' And the plan never came, and...I don't know. You really need leadership to push...to push that as well. Certain things. Really that should come out of the group [though]

(X.4.40)

We recorded the results of the discussion on the flip chart, and also some to-do's, and we put that into the kitchen. Because, also thinking that everyone goes into the kitchen from time to time to get coffee or tea, but also when you wanted to smoke you had to go through the kitchen into the outside. So that was very much perfect in the kitchen. But somehow that didn't work. The very first thing you saw when you entered the kitchen was the flip chart there...but no-one seemed to noticed. At the next meeting we would

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

look at the to-do's and...'no I had forgot.' And 'I didn't have time' or, 'I said I would do it? Oh yeah, I didn't remember.' I am not quite sure why that happened. Perhaps they are concentrating on something, or perhaps...I really don't know.

(X.3.21)

The reason for this type of inaction in the team environment seems highly related to team expectation and accountability. One interesting factor that was not related to a specific agile practice, but that was present in all agile teams, was the expectation for action resulting from discussion:

One of the interesting things that she noticed was that a lot of the other groups that she had been to visit were going on and on and on about quality all the time. They'd talk about it. We didn't talk about it. We just did it. You know. We didn't have to talk about it

Was there a reason for that other than just having good people?

I think to some extent that was our picture when we went in. No...you know, we weren't prepared to leave bugs behind. We weren't prepared to settle. But you know compared to most projects, where you end up in the horrible panic. We weren't prepared to let things decay.

(L.2.8)

The flip side to the pairing thing is that if you are not pairing effectively then you might spend an overly long amount of time discussing the pros and cons of a certain approach vs. another approach, when you could have just gone ahead and done it. So you kind of need to keep an eye on...you need to have some discussion about how to do things, but you need to at some point draw a line and say, ' ok, we've both agreed that we will do it this way, or that way. Just pick one' It's just that analyzing the thing to death is not always productive.

(L.3.28)

Indeed, the social support that comes from the agile environment in terms of action is very powerful:

So I am trying, like with information radiators. Some days I decide to say, just, 'Why talk about it, just do it.' And I am trying to get more of that. And actually we have, ah, currently seem to have some success in fostering that kind of behavior team wide. So we had one team member who was very, conservative about process. And also very pessimistic. And he wasn't bad at complaining. And saying 'Oh that doesn't work for me.' And I am pretty good at doing that too...saying 'That doesn't work for me' and then sitting back and waiting for things to become different. Recently, one of the things that he complained about for years now, is that he wanted to do more pair programming. And you know, in every reflection meeting he said...'We didn't do enough pair programming,' and that was it. And now he finally...I'm not quite sure how it happened...might have

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

been that someone said 'Just do it! Do something about it.' And he actually seems to have thought about it, and after one or two weeks he came up with, 'Lets do a pair programming matrix.' That's what he came up with...And now he's actively taking care of it and pushing it forward. And that's a great experience. Always sometimes also a little bit annoying to me, personally. But it's great to see him taking action. And feeling responsible for the process. I think that since then, around the time that happened. The whole team culture has changed. So it happens more and more in the small. And so people just do things, do the things that are necessary to be done. Instead of waiting for others to do it.

(X.3.61-64)

Retrospectives

Retrospectives were a practice that was strongly related to highly cohesive teams in this study. Retrospectives were team meetings conducted to evaluate how the team worked together to develop software:

Everyone on team, we have regular meetings to discuss how everything is working for us, you know, aside from all the features and that, we have what we call a postmortem. Every five weeks or longer, we discuss what's working for the team and what isn't, and most of the, the core of things that we are doing are working well for everybody. And if they are not working then we usually address it.

(T.3.25)

The topic of retrospectives differed between a sole focus on process issues to a more general focus including team communication and working environment. Participants associated regular retrospectives with a strong team feeling and togetherness, and such meetings were seen to play an essential role, both in increasing solidarity and in effecting change in the team environment:

Retrospectives, I think those are really valuable for trying to make everyone feel like they own a problem and they can actually solve that.

(L.4.8)

You know there is things like you write on the post-it the good things that happened in this week. Which is also very, very good to remember because you sort of forget to say - 'Actually we worked really well as a team.' You know at the end of an iteration you forget things like that or...you know because a lot of the times people like talking about the things that peeve them and then they forget that actually some things were good about that iteration...And in addition to writing the good things, you sort of write down about the things that are negative about the project, things that didn't go too well for whatever reason. It might be people. It might be environment. It might be like, 'You know it was hot in the room' to, "Somebody's piece of code fundamentally...' Makes the point, you know...and you can't put that up without suggesting an action to take from that.

(L.4.10-11)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Part of the value of retrospectives seems to be in their ability to strongly affirm team sentiment regarding the project and team environment; in other words, it often became very clear during the course of the retrospective whether other team members had bought into team goals, and what their motivations were:

> *So that's I think where I really like the benefit of the retrospectives... basically its about you know eliciting people's feelings about how the project is going and therefore, you know, what is slowing it down, what are the things that are really good because you don't want to throw away the good stuff as well.*
>
> (L.4.14)

Particularly important was the knowledge of team feeling that came out of these meetings:

> *You can get a sense if people are happy in the meetings, happy with their work or not in the meetings you know. People don't grumble under their breath.*
>
> (T.3.41)

Participants in highly cohesive teams seemed very happy with, and in some instances, quite proud of their functional team environments:

> *It's, everyone on the team right now has adjusted quite well to working this way and; in these meetings that we have to assess how it's working for us nobody ever says that we should stop doing it. Yeah no one ever says that. Its, from my sense everyone appreciates the benefits of the process.*
>
> (T.3.42)

Retrospectives additionally increased awareness of the team as a whole – how it was working together, and what corrective action could be taken improve the team environment and 'get everyone on board'. This seemed to be associated with increased identification with the team, and appreciation of the team environment as something valuable and worth investing in. Retrospectives were highly related in this study to trust, respect, positive feeling, and buy-in to team goals. Much of the strength of retrospectives discussed in this seemed to be that they occurred every week. Problems would be discussed and action plans would be made for action. The following week the whole team would follow up on the progress made towards team goals firmly establishing team expectations and accountability for action.

Connectedness

Having whole team awareness was also highly related to feelings of comfort and security in the team environment. Participants in cohesive teams were seen to possess a feeling of mastery on a team level. Team members with a good grasp of team activity and a certainty regarding the actions of others in the team were more likely to feel secure as 'part of the team,' and be motivated to invest in team efforts. Feelings of connectedness to other members of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

team also seemed to be associated with increased feelings of excitement, involvement, and motivation.

Awareness of Whole Team Interaction

The agile practice of having the whole team in a shared team room was seen to be strongly related to feelings of connectedness in the team environment. Team members were not only able to communicate with others more effectively, but were additionally aware of the *lack* of communication between parts of the team:

> *I mean at one point we felt that we had no real, you know, it was ridiculous that we had the business in the room but we had no communication with the business whatsoever. And possibly not even with the business analyst, and you know the business analyst said well actually it's funny but the business also feels exactly the same way. So we changed the process so that we didn't write everything upfront, we didn't write the acceptance tests upfront. And that meant that you have to go into the business and get those acceptance criteria which is a good skill to have I think. So you know and it's also good because you can talk to the business, and you know more people who talk to each the more, I mean there is also room for confusion, but there is also like a feeling of communication.*
>
> (L.4.78)

Another interesting phenomenon present in several cohesive agile teams was the fact that all team members would always show up to non-mandatory meetings. Part of the reason for this was that if team members did not show up, those who did attended would pester them once they got back to the team room until they did. Presence in a shared team room increased the chances that others would notice absentees in shared communications and would be able to follow up with them about it:

> *There is somebody who recently joined our team...and for a long time he wasn't, for the first two weeks he was not coming to the daily standup meeting. And I distinctly felt like...I didn't know what the heck he was doing. He sat just beside me! And I didn't know what the heck he was working on and what current task he was doing and I did end up, I think I asked him a couple of times you know 'What are you doing?' and 'What are you working on?' and 'How is it going?' Because I kind of knew his general area of expertise but I didn't know what he was doing...then I started bugging him about not coming to the daily standup meetings. So I think so now he comes but you just don't get, I think you just don't get the sense of what someone is doing unless you directly ask. And even though if I ask him what he is doing he will tell me. But nobody else will know what he is ... I think people will get a sense, that they don't know what he is doing either. So I find that...disconcerting if I don't know what everyone else is doing. In a general sense, you know.*
>
> (T.2.39)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Such whole team interaction was seen to greatly increase feelings of motivation. This motivation can be partially attributed to knowledge of the importance of the project in the eyes of others:

> *So all of those people came everyday to the Scrums and talked and actually even the marketing person came to these Scrums and talked. So I think what I liked about it was the enthusiasm I guess that everybody had for the project. Having those people show up everyday and having people, you know just really enthusiastic about what we are doing, what we are trying to achieve and all really backing the project*
>
> *(T.4.1)*

Keeping a Finger on the Pulse

The agile team environment allows team members to easily keep in touch with the activity occurring with the rest of the team. Working in a shared space increases awareness of team activity and discussion, and team members remain relatively 'in tune' with the rest of the development team. Involvement in the 'heart' of a project as allowed by a shared team room was seen to help team members maintain a sense of security in that they know what is going on in the project, and that they are involved in the communication that they need to be involved in. This also helps maintain a big picture understanding of the software project.

> *Like I might be working on my own little bit of the job. But I will kind of be aware of what the guy next to me is doing as well, and what the guys across the other side of the corridors are doing. Because you are right there and you hear the discussions. You are not actually involved in the discussions but you have got an opportunity to be involved in them. If someone starts talking about something and they start talking about something that is related to what you are doing, then you can prick your ears up and you can join in the conversation if necessary. Whereas if they are on a different floor, then you are never aware that that conversation has taken place, So they might be working on something that effects you but you didn't know that. It's good to just...it's much easier to just keep your ear to the ground and get a good picture of what is going on when everyone is sitting together.*
>
> ...
>
> ***And you don't find that having these conversations going on around you is distracting?***
>
> *Ah strangely enough no.*
>
> (L.3.22)

Daily team meetings, especially, were seen to allow team members to maintain an awareness of the project. This awareness was related, not only to what was currently going on in the team, but to future events:

> *So they will give us an idea of when they are available as a resource or not or you know things of interest that they have done. Usability also they give us heads-up about what they are going to be starting to investigate, any usability testing that's going to*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

come up; whatever they are blocked on. Usually it means they will say something like 'Oh well I will have to talk with you. If you plan on...if you are going to be programming that, and so we need to talk.' And then on the development side we just, I think it's mainly to...the types of things we talk about or whether there is a new feature appearing in the program or something that we changed that should not affect what's happening, but if people start seeing errors, problems in a certain area, then they know who to call. Anything else...and then we have more product management or product specialist type folks who usually mention if they've recently completed a trip to visit a customer or to some sort of seller/contractor or whatever, I don't know. Then they give us an update.

(T.2.27)

Team members in non-agile teams were much more likely to find themselves 'out of the loop' with regards to team activity. The feeling of not knowing what is going on and somehow missing out on important information also seemed related to lack of initiative in the team environment, and lack of team awareness was also related to a lack of excitement and involvement in the team and project:

I was recently working with a non-Agile team and I couldn't believe how much I did not feel a part of the team. I didn't understand where the information was coming from - they didn't have scrums. You know, there was some flow of information but it was really hard to figure out where it was...I mean was it email that I wasn't getting? Was it a meeting that I didn't know about? Like there seemed to be this like weird way of getting information that I had to try to squirrel and figure out before I could get in on the stuff. But I never really felt I was completely part of the team. And I talked to other people and they didn't either. It was like, it was a larger group. So it never really gelled as everyone's working towards the same thing. It was a bunch of people working towards different things.

(T.4.38)

Team members who were not in shared team rooms noted the importance of keeping in touch with others several times a day. This not only increased team communication, but seemed to help maintain an awareness of others on the project and increase the amount of sharing and participation surrounding the tasks of individuals in the team:

Because I was on a different floor what I would do is make sure I walked up through there during the afternoon. So just like you know just a quick walk through and allow the developers to say oh since you are here I have a question for you. And I found that was very important because I wasn't sitting there and I had the other interaction designer doing that as well. We just two kind of would walk through and let people flag us down if they wanted to or somebody would say oh I have got something really cool to show and they would bring us over and show us a prototype. So, that I think that really helped an awful lot. Like I don't think you could just do the Scrums.

(T.4.9)

The ability to know that the project was moving along smoothly and that everything was on track was related to reduced stress and increased security and motivation. Some participants mentioned the increased awareness in the face-to-face environment regarding unspoken problems or issues:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

And you could always tell when there was a problem. It was like somebody sweating in the room somehow. You can just like, smell it. If people are scared they sweat. It smells like ammonia almost...and you could smell it in the room and I could walk through the room and find it.

(X.2.24)

While it was difficult to confirm this with the data collected in this study, the previous quote suggests that there is often a physical location for tension in a team room. In a shared team room, it is relatively easy for team members to identify the source of the problem, and it is much more likely that they will be aware of the nature of the problem and offer help if they can. If the team is not in a shared team room, problems like this still seem to present themselves as a vague sense of uneasiness picked up through team interactions, but it is much less likely that that the problem can be 'sniffed out' and resolved.

Overall, participants showed a strong preference to being highly aware of team activity such as is allowed in the agile environment. Such awareness of the rest of the team was related to increased involvement, and was associated with higher levels of helping, trust, and goodwill in the team environment:

Do you feel when you are doing your work any dependency from the team?

Yes. But that is not any different between agile and non-agile projects. You can have a situation where the team is depending on you at any time. The component is not going to work until my bit is done. So even if they finished with their tasks yesterday and I wasn't going to get my bit done until the end of this week, then they are really depending on me to get my bit done. It's actually I think, accentuated on a non-agile team that dependency. Because on an agile team, if my bit didn't look like being finished until the end of the week, then um, then they wouldn't be twiddling their thumbs waiting for me to finish it. They would be like "What can I do to help?' And people still do that on non-agile teams, people are not totally siloed. But it's just like breathing on an agile team. that's just like, that's the normal thing. Whereas on a non-agile team it can be like "Well I'm helping you out here; I'm doing you a favor.' And you are like 'No you are not. We are both doing the same number of hours on this. We are both aiming for the same goal. We are both just doing our jobs.

(L.3.17)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Individual in the Team Environment

Equal Partnership

One of the major characteristics of cohesive agile teams in this study was the absence of command and control relationships, and the presence of equal partnerships in the team environment. Leadership tended to be more flexible depending on who was most knowledgeable regarding the issue at hand, and team members were more likely to view each other as equals; meaning that while one person may have more expertise in a certain area, others were still entitled to an opinion regarding any part of the project. When there was a project manager involved, he or she was more likely to provide support, guidance, and direction rather than no-questions-asked directives. In such teams, individuals seemed much more aware of, and willing to engage with, inefficiencies or problems in the team environment. Agile practices strongly encourage this type of 'self-regulating,' or 'self managing' team environment and behavior, which also partially arises from management and worker interactions:

> *It was kind of hard to motivate these people because it was so different from what they were used to... these guys were used to, they would ask for a task and you would give it to them and they would go away and work for a little while. And then they would come back and ask for the next task. Which was, um, not exactly self-managing. So trying to get these people to actually be self motivated themselves was kind of hard at first. But then they kind of picked it up and ran with it after I started closing my eyes and pointing to the screen every time the came to ask for something. Then they realized that they could pick any item of the list that they wanted to. So if they wanted to they could pick the easy ones. If they wanted to they could pick the ones that looked like they might be interesting to solve. They could do whatever they liked. So once they realized that then they were off and they were working away... the initial resistance was interesting. But after that it was quite...really rewarding, they could pick it up quite nicely.*
>
> (L.3.3)

The empowerment expressed by individuals in self-regulating teams can be compared to a sense of powerlessness present in individuals operating in environments with strong hierarchical structures, whether formal or informal in nature. Such individuals showed a strong tendency to defer to those in power and seemed dissociated from taking ownership, responsibility, thought, or action for team occurrences outside of their specified roles. Teams able to instantiate this 'freedom' in the team environment and taking of responsibility, were seen in this study to have an increased amount of energy and enthusiasm on the part of both the team members and management:

> *The previous team I worked on, in the environment they traditionally had the architect on their own you know, all that kind of stuff. And we did the XP thing, you know, with just a regular bunch of programmers. And when we first started the stand-ups, the feedback sessions were a bit, ah, stiff. Because no-one would say anything. They'd just sit there. And by the time we were finished, when we finally got them to realize that it was their call - this stuff, lots of stuff - and they were expected to put out opinions; you couldn't stop 'em. And that was a...and the hardest thing with that was*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

getting them into the project. And it worked in spades, in that case. I mean, I wasn't there for the end of it, but they completely lit up; compared to how they used to be. So I was quite surprised with that one.

(L.2.25)

When team members are all more or less on the same level as opposed to being accustomed to automatically deferring to authority, there some valuable interactions can occur, as will be discussed in the following sections.

Buy-in

Importance of a Shared Goal

One of the most important aspects of cohesive teams illustrated by this study was buy-in to the group goal:

Have you ever been on a team where people have not been responsible or reasonable about their work or, for example, in estimating tasks?

Ah. Not when it is run in a collective fashion like that. In situations where it is not run in a collective manner, where it is run with the team leader telling them what's what, then that tends to...you don't get quite the buy-in for the team that way. It's all about what your goal is for the team. And in an agile team it's all about producing the most business value for the company that you are working on. And on a non-agile team, and this is not always true because there are non-agile teams that have good teamwork as well. But what tends to happen is that it is much about your individual work. So you want to say "I did all of this work. Look how good I was, I stayed until midnight every night and I wrote half the code in the application.' So you get this kind of hero programmer mentality. And that is detrimental to good teamwork because everyone is like 'I want to show how good I am' And they are kind of competing with you rather than working with you sort of thing.

(L.3.14)

It really depends on the group that you have at hand if it works together. Although maybe when all people really want...want it to work, then it is a lot easier. Because we have this one person that is not really wanting, and you have to control, you have to say what he should do, and the other people in the team get pissed because they always have to solve the problems because he only did his small thing and is not able to solve other problems

(X.4.43)

In addition, it seemed especially important that team members exhibited *a willingness to work together* to achieve this goal.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An agile environment that allows relative freedom and flexibility in tasks and roles is suited to teams who work in a collective manner, and the effectiveness and speed that has been associated with agile software development teams relies on these kind of interactions:

> *If it's just about everyone doing their own little tasks, then it's very easy to, sort of, you can slack off a bit. And if someone has given you a task to do and their estimates are slightly wrong then it's very easy to, if they are quite generous with the time allowed then it is quite easy to make the task take up the time. So, whereas, as soon as you get to the point where your task is done on a team where you are working more collectively ...It's a difference between you having a task assigned to you and then once you have finished that you have another task assigned to you, then you've got a big, everyone had got a backlog of work. But the whole team has a backlog of work. And that backlog becomes my job to do, and as soon as I have finished this I will just; there is another thing to do right there. So there is no slacking around at the end of having completed a task because there is always like, the next thing to do.*
>
> (L.3.16)

Having both buy-in to team goals and a willingness to work towards them was seen to result in helping behaviors increased the team reliance and solidarity discussed in previous sections:

> *You try and work with [non-agile team members] as a team and you try and get them together to discuss something surrounding the component we are working on. And I find that they are only interested in working on what they need to get done personally to get their bit done. Their goal is not to make the component to produce business value. Their goal is to finish their task. Whereas on an agile team if I wanted to discuss what was the next step to get this component working, then everyone would be more than willing to come on board and discuss it and work; sort of, everyone cares about - for example if I was finished the bit I was trying to do and you were finished the bit you were trying to do, then I wouldn't even hesitate I would just try and immediately find something that would help you finish. And it wouldn't really be your task and my task, it would just be – these are the things that need to get done and we have to do them. Whereas in the team I'm in at the moment it's very much this is your task, this is my task.*
>
> ***Does that effect how you interact with the other developers?***
>
> *Oh yes definitely. I found it much easier to keep up that enthusiasm, not just for the job, but I'd be far more willing to put in more effort if the whole team is all working toward the same goal, you know.*
>
> (L.3.15)

Whereas if personal goals overrode team goals, team members were much less willing to help each other:

> *At one point they started to say 'Okay, how much time do you think you need, and you have to commit to that.' And they did that to the other team members as well, and so there was four guys, having their personal goals, so they were not willing to help each*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

other anymore. Because if they didn't get to their personal goals, then they were hammered by management, saying 'Hey you committed to this goal, and...' So the little bit of team spirit that was still left was killed as well

(X.4.47)

At that point the bugs aren't shared. They don't seem like a shared responsibility to fix. And so I am more reluctant to ask someone to help me on fix a bug because I know that he has his own bugs, as well, to work on. And you know maybe he is less likely to offer help because he has his whole area of bugs as well. And so because they are parceled off to each individual I think it's harder to force someone to help, come and help on your bugs and likewise share bugs. But when it's at a feature level and like its some more larger area of code then I think it's easier to draw someone in and have them share an area of code as opposed to individual bugs. Which is too bad. Because I have some bugs I need help on <laughs>.

(T.2.8)

Situations where team members were asked to do tasks that would inconvenience them in relation to their goals, or when they were asked to do something that they do not feel that they should have to do, were strongly related to 'push back' behaviors or righteous passive aggressiveness:

Well the administrator comes to me and says "Well he orders me around and treats me like a slave, I said we will fix that. So I said to him "Could you do these yourself please. Could you do these for you." And she has learned to push back. And say 'it's not useful for me to do this task, it's useful for you guys to do this task. I am not learning anything and you might. You know you will know better, you will know something from doing it, not me, I am not learning anything from it.' And that's very interesting and you get back on pushback, I love that stuff.

(X.2.103)

This study clearly illustrated the value of avoiding conflict between personal goals and team goals, and therefore encouraging teamwork and momentum by allowing people to work together towards the same goal. The question remained, however - what is it about the agile environment that helps to generate buy-in to the team goal? Firstly, the presence of a team goal is a strong part of agile culture and seen as essential to proper implementation of agile practices. Managers instantiating cohesive agile teams, therefore, were seen to establish a firm goal and a collective culture surrounding it:

The training that we took for Agile that was very important that not only did we do this but everyone agreed on it before we started programming because otherwise this person's trying to make this application, this person's trying to make this application and you are going to get conflict. Whereas most of our conflict came in, we both want to get here and we disagree on how to get there and so but we were still trying to do the same thing. So you know it's not like we are trying to, it's not like we don't believe what the other person's trying to achieve, its that we are just kind of arguing over how we achieve that. And that's a lot easier than if you have someone saying okay I want X, I want Y and it's completely separate.

(T.4.44)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

There were a number of other motivators to buy-in to the team goal existing in the agile software development environment however, as will be related in the following sections.

Taking Responsibility

The allocation of responsibility in the agile team environment was seen to have an important effect on agile team members. In many agile teams, developers are given the freedom to choose and estimate the stories or features that they would like to work on for the current iteration. This seems to encourage, to a large degree, buy-in and initiative in the team environment:

> ***What is it about being able to pick your own stories that is motivating?***
>
> *It's about buy-in. So if I get handed a task and they say 'can you do this task please and you need to have it done by the end of the week' then I will try and do that. But if I haven't estimated that myself and there is a tight deadline I'm not going to take the responsibility to go and actually meet that deadline, because it wasn't me that came up with those sorts of timeframes. But if I came up with a timeframe myself, and estimated it myself, then I kind of feel obliged to do what I said I'd do. And also, you get to choose what interests you. You get to work on what you want to work on, Rather than what they tell you to work on. It's all about buying in to the overall goal, rather than this is my task, this is your task.*
>
> (L.3.4)

Thus team members who had more input into the processes occurring in the software development environment were seen to feel more in control of events, and be more likely to feel a sense of ownership and responsibility for their tasks and their place within the overall team goal:

> *I think its empowered a lot of people to help and want to help one another whereas before you didn't have to go an extra mile because your authority was only this big and when you have done that, you are complete, you didn't have to go above and beyond to be measured as a good or as an effective employee.*
>
> (I.1.15)

Agile environments were seen to increase the amount of freedom and choice for individuals while simultaneously providing the balancing factor of assumed responsibility and accountability for individual decisions and actions.

One of the interesting factors present in cohesive agile environments in this study was that they would, where possible, allow for the allocation of tasks based on interest as opposed to skill. Practices that allow for this include: pair programming, which supports teaching and learning between developers; test driven development, which ensures that a developer is much less likely to cause problems in the software environment; and the allocation of tasks in story chunks for a given iteration, meaning developers could easily move back to another area of development if they cannot deliver regarding new types of tasks. Therefore agile methodologies

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

were seen to increases the chances that individuals would get to work on tasks that they would *enjoy:*

> *It's sort of changed a bit. I mentioned that we all have sort of areas of expertise. So at the beginning we were little smaller and the areas of expertise hadn't really been created yet. So we would often have mixed up more regularly of who worked on what so that everybody got exposed to areas. But we found that over the months that people did have a personal preference on types of things they liked to work on.*
>
> (T.3.38)

The fact that team members could choose their tasks was seen to allow them to choose tasks that they were good at, and that they could get done within reasonable effort; thus avoiding situations where a developer delays completing a task because he/she has no idea how to start working on it, because it is beyond his/her capability to complete alone, or because it is a boring tedious task that he/she is not looking forward to.

> *I think it's really important that they can focus on the work that they like...because there is lots of stuff that they wouldn't be experts in doing. So it makes sense for other people to do these things. To have to set up the architecture, and have to write the documentation, and have to talk to the customer. Because otherwise they just put it off and procrastinate and don't do it. And so I get my assistant to get the information for the documentation. And if you are interviewed by a pretty girl, you know, it's just different from having to do <makes typing gestures>. It's just different <laughs>... What really helps us is the education system...the young guys, they do all the mundane stuff that is boring for the old guys, but it's really fun for them. They are happy just to do anything with computers, and they are really fun to work with because they are so full of energy and are so excited.*
>
> (O.2.9)

A balancing factor to the fact that team members are able to choose their own stories was seen to be the increased awareness and association with the rest of the development team in the agile environment (further discussed in the section on *Recognition and Leadership*). One participant recalls:

> *So he had a question with me other day about collaboration... he said 'Well if there is a task, if that someone naturally knows how to do in the team doesn't it fall to them?' I said 'no, the people have to volunteer and if they don't want to do it ... and nobody else wants to do it, we have to discuss, how we are going to solve that problem as a team'*
>
> (X.2.103)

In this case, a personal problem becomes a team problem. If one developer cannot do a story, then someone else will have to do it. This understanding in the agile team was seen to result in the fact that tasks that need doing get done:

> *If people see the stuff that needs doing then they will do it. But also they will choose to do the stuff they prefer to do.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(L.1.90)

The shared team environment was also seen to allow respect and appreciation towards those who put in extra effort, or those who were stuck with especially difficult tasks. Overall, the freedom and choice that individuals have *within team constraints* seems essential to motivation and enjoyment:

So what is it you particularly like about the team environment you are in?

Yup. Um. The most important thing is just the lack of the bad things that most software teams have. Of pushing people to do things, rather than encouraging them to do it. So it's kind of cultural. Working out what needs to be done...and getting....letting people choose to do it or not.

(L.1.88)

The right people in the right place

The software development environment is a highly complex arena, with many dependencies present and interactions that must occur in order for software to be built. For example, one team member describes his team environment:

I would define our team as the entire group including the, you know, person in QA, the person in usability, the product designers, the managers. That to me is the entire team and then within that you have your development team, your programmers, but there's always a Venn diagram, you know. If you have got two, one or two or three developers that are maybe working on a particular feature, depending on the size and complexity.

But then you have got a little circle around them plus the one or both of the usability people and you have got probably one of the three or four feature owners which are one of the product specialists.

So, you'll work with other developer, and those other two or three people to define the feature, how it's going to look, how it's going to behave. So, you've got usability working on the behavior, your product specialists trying to define the feature, and then you have got the developer in that mix of developers trying to provide the information required to, you know, tell them that that feature is going to take three man-years or this is what we can do; you give the realistic information on what can be done in the time allowed, you know, a closer understanding of the constraints.

So, between those three little people within that little small team within the big team, you're working on the feature together. For example this morning I had a meeting just like that; where there is a couple of features that I need to estimate, but I needed more information on what the feature is exactly going to do, how it's going to do it, how usability wants it to work to make the user successful.

So we were in a meeting this morning just trying to figure all that out. And until some of that can be done I can't really estimate how long it will take, and then how long it would take affects the overall schedule that if we now have a task it's going to take six

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

weeks but we have only got two weeks left in the cycle, either have to start breaking it down or pushing it forward and in the big picture on the whiteboard. So we try to do all that at the beginning of the cycle when it's still somewhat malleable.

(T.3.15)

Such a complex environment is highly suited to, and perhaps even requires, the kind of self-regulating team behavior supported by the agile environment. Agile team members can easily draw into communication one, or two, five colleagues; whoever is has a stake or is interested in the current issue. Because you are in a shared team environment, the chances that people who should be there involve themselves is increasingly high, so there is less of a need or a to drag everyone in the project into a room in order to be sure that nothing is being overlooked:

And I mean, again, it's only the relevant ones for, which you can feature that are involved, so that's nice to because you're not sucking up the time of the whole team to listen to things that they may not be working on.

(T.5.12)

Relevancy and involvement were common themes in discussions with study participants. Communication that included interested parties, and did not include those who were not interested, was related to a much higher level of enjoyment and energy in the team environment:

I started doing interaction design because I am interested in it. Developers are doing development because they are interested in it. If they are not interested in listening to a usability test result, we would make them not mandatory so people could leave if they wanted to right. If they are not interested I don't want them sitting there because they are not going to care, right. But I do want them to accept our results if they are not going to sit there still. So we are going to be talking anyhow at some point but they don't have to sit through the whole thing and I don't hold that against them. I mean I don't want to sit through being told, you know, a performance tweaking study or something because the developers are interested in that.

(T.4.13)

One of the major factors, however, seemed to be that interested parties *could* include themselves in discussions if they wanted to and were not busy at that time, even if it was an area of the project that they were not 'officially' involved in. This interest-based participation seemed to give team members more of a sense of freedom in the team environment, and allow them to take ownership of the activities that they were involved in.

Individual Recognition and Leadership

Agile practices were seen to create a shared environment where individual contributions can really shine out. This was seen to greatly encourage team members to put more effort into contributing to team goals:

In a project environment, the team is doing stories but there are issues that don't have to do with design. Things that need to be done that are just hanging around in the air. And they are waiting for someone to pluck them out. Because, the project manager doesn't necessarily know what is going on, you know. And one person could do that, but

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

you know, you all have this work to do...in a team environment someone who does something will get thanks and recognition from the rest of the team. But some environments just don't work this way. You just keep doing your own work.

(O.2.6)

One participant described the effect of adding an information radiator that highlighted individual contribution in an agile team environment:

The ritual involved in, having written a test, standing up and going over to the whiteboard, and looking around that, a (keeping?) that everyone see what you are doing. And getting, and making your number of strokes, and getting praise for it from the rest of the team, and everyone else feeling that now they have to, ah, have to keep up writing tests. So that they don't fall behind.

(X.3.2)

Other agile teams used other indicators of success, such as toys that make sound, or marks on the project board; usually it was something quirky or fun. The important thing seemed to be the fact that everyone notices what is happening and gives recognition:

And that green dot is like an endorphin rush. It's like, woohoo. But the code has to be checked by an external person of course, but once it's check and then you can put that up there, and it's great.

(O.3.16)

The appreciation of others on the team for day-to-day efforts, combined with the ability and initiative to fully contribute to the team environment was shown to be immensely powerful in increasing individual satisfaction:

It was very interesting because one of the women on the team, we didn't need her full time after a while, just needed her part time. And she had other work that was being paid, oh my god, 50% may be even 75% more. And when we ask her to come back and help us part time, she took the old wage. She just wanted to be a part of it. Because of the team.

Because of the team?

Because of the team; she loved working with us, with those people, she could do what she wanted. She loved environment, she was...her say, she was the pro in that team and is considered the best, not the best programmer but the best at what she did. And she said sure. That's very powerful. And that making a team so they want to be there not have to be there. And they want to make it happen, not to have to make it happen. And I think that's key.

(X.2.77)

Agile team members working in a shared team environment were seen as more likely to be recognized for their work and individual efforts. In addition, they were more likely to be encouraged to bring their expertise and experience to the table strive to improve and expand on

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

their existing skills. This can be compared to environments where members are not recognized for their expertise and/or don't have the opportunity to bring it into their daily work:

> *It's very difficult because most people are used to command and control, right? So when they come on board they want to be told what to do. And the transition for them is very uncomfortable. They have to begin to learn and test the limits. So in the past if they try and speak out they would be told 'What are you doing? That's not it, do this and don't bother me with that.' They would be humiliated. They would be you know, treated like crap.*
>
> (X.2.41)

> *You can't fix it, if they pay you enough, and that's quite frustrating. Because it means your opinion as a specialist, and with experience, who has good ideas - but you don't even have a chance to do anything with it. It might mean that I should be looking for another organization.*
>
> *(X.4.35)*

Presence of Goals

Finally, the agile team environment was seen to increase the presence of team goals in the project environment. Constant reminders of goals were seen to be essential in getting team members to really take them to heart. Firstly, the visibility of goals was increased though the use of information radiators:

> ***Do you know if information radiators have some kind of effect on the 'flow' of the team?***
>
> *I think one of the most important things in the team is a common goal. And information radiators can help with that. Not making a team goal, but making it visible. Making the team aware of it. So what we recently started to do just two weeks ago, is that at iteration plan we are working on parts, and what we found is that we have the best flows when we have a team goal. When we are all working towards this common purpose, and because we are working for a number of customers at the same time we often don't have that. Some people are working on one part of the system, and others on other parts, and ah, actually it's to...there isn't much sense of team. So, but there are...part of the team will work together, and the other part will work on something else and there isn't that much communication. And so...and not only communication, but also, they don't know what the others are doing, and how well they are doing, or whether they could help, and yeah...collaboration is not very good. People working on very different parts of the project, and for different customers. And what we noticed, is, especially in times of very high pressure. Of very high deadlines. We often managed to overcome these problem working as one team, and even working for different customers, having a sense of having to, um, meet those different goals as one team, and having to go there together*
>
> (X.3.40)

Having clearly visible goals was seen to increase the focus in development teams, and the likelihood that individuals would work together towards goals. Increased social interaction in the agile team environment was also seen to help entrench team goals in the minds of agile team members.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Communication and Collaboration

One of the main effects associated with agile software development is increased communication and collaboration:

> *In my experience the big thing about agile software development is collaboration. People talk about shorter iterations and feedback loops and things like that; which are all true. But you can have non-agile processes with much shorter iterations. In the old days computer programmers used to be just geeks who are good at code and you can't do that any more and be effective on an agile team. You must have collaboration skills. That the additional element that I think agile brings to the table. You tend to achieve much more synergy in the team.*
>
> (L.3.31)

This section explores the characteristics of the agile environment that encourage close communication and collaboration, and discusses the effects of such close interaction on team members.

Ease of Interaction

The agile environment was seen to make it very easy to interact with others. Firstly, all members of the team are sitting in a shared team room, which makes it a matter of shouting over your shoulder, calling someone over, or walking a couple of paces in order to engage in discussion or have questions answered. One of the main instigators of communication is the daily stand-up meeting where each team member is required to report their daily progress and any problems that they are having. This information is likely to lead to further communication on an individual basis after the meeting has been closed:

> *We have our daily meeting, and then typically what will happen after that is, you know, we'll get, or me, I will get flagged by a developer who is working on a feature that I am working on and they will talk to me about any question they might or they will show me what they have done since yesterday. And then, you know, we often communicate by email as well.*
>
> (T.5.9)

> *The daily meetings we have running 15 minutes long where you get around the room and they start discussing the product they are working on. At the end of 15 minutes half of the team is still in there working together on laptops because they have heard about the problems and they want to help each other out. So that team meeting in that room becomes that little sort of common area for a while and then the problems all get solved or there is work items everyone can do individually and then they scatter and disperse back into their cubicles. But it's a start.*
>
> (I.1.16)

An important aspect of the agile team environment to increasing communication and collaboration was seen to be *transparency*. Many of the agile practices are set up so that all activity is observable or transparent to others, and openness and honesty were seen to be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

encouraged ubiquitously in cohesive agile teams, from developers to management. It is expected that problems, issues, points of contention, and points of interest, will be brought up immediately and discussed. As one agile team leader noted:

This isn't a place that you go and hide.
(O.2.14)

All roles are involved throughout an agile project, and are readily on hand for quick and easy discussion of issues. Involvement and availability are generally held to be important, where team members are available and open to be asked questions or drawn into discussions.

> *Throughout the project, whenever you think that you need to find anything out at all you just go and talk to people. That's another thing that is very different in agile, is that all roles are involved throughout the entire process.*
>
> (L.3.20)

This was seen to greatly increase the speed of communication and momentum of projects, and seemed to make agile team members much happier in their working environment:

> *Well, for that session it's essential to have all voices present. In the past that might not have always been the case on other projects, it might be the developers and you need a lot of input from the expert user or the product designer and then it bogs down the speed of communication because you know you grab that person or you have to wait a day until you can get their input.*
>
> (T.3.9)

Some of the major barriers to communication seen to exist in the working environment were related to social rather than physical impediments. The barriers most prominent in this study were: the reluctance of team members to waste either their time or the time of someone else in non-productive conversation, and the reluctance to interrupt others. On a pragmatic level, visibility of others in the shared team room was seen to allow team members to assess whether the person they would like to talk to is perhaps concentrating on a problem, or talking to someone else, or in a bad mood and should not be approached at that time. Individuals can then wait for a mutually beneficial time for interaction and grasp it. This can be compared to a more dispersed team environment, where individuals would likely have to interrupt both their work and the work of others in order to meet to discuss an issue.

Interaction in the agile team environment was also seen to allow individuals to take advantage of opportunities for collaboration, rather than only interacting with other team members when they become stuck on a problem. Daily awareness of the progress and concerns of other team members, in addition to and understanding of what they have worked on in the past, was seen to allow team members to make connections between their work and the work of others. This was related to increased helping behaviors in the team environment, where individuals will offer their knowledge and expertise to a problem that someone else is having:

> ***Do you find that knowing what other people are working on, even if it may not relate directly to what you are doing, changes the way that you work?***

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I don't know if it changes the way I work, I mean, but it helps me to be more, you know, efficient. Or helps me, like if something I am working on might be related or might be similar in workflow or have any kind of similarities then you know you can go talk to them, maybe they've already solved the problem or have solved it in a way that you can leverage, but it wouldn't change how I worked on my feature, it's just enables me to, I don't know, be more effective at being successful in my, you know, you might be able to share some ideas or code or who knows what, some brainstorming. I would go and to talk to someone on the team, I was going to do it this way and say you know I thought it would probably work for I am doing too, and then you realize that you can create some code that you can both use -- your features on top of it. I would say it helps in the collaboration or the architecture of your code.

(T.3.24)

The presence of the institution of pair programming was seen as invaluable in this regard, since it becomes very natural for the person offering help to become the pair of the person in need of help. Helping behaviors such as in the previous quote were seen to be essential in the software development environment, where the knowledge to solve problems is generally dispersed across the team, and where asking for help from other team members can greatly reduce the amount of time that an individual spends on a problem:

There was one instance I remember where the pair programming really helped where I had one area of expertise and I got someone else for the different area of expertise and we had to combine what we knew to actually get the final result because otherwise it would have taken me 2 or 3 times as long to do it on my own. Or if I gave it to [my pair] it would have taken him 2 or 3 times as long. But together we were able to get it in a shorter amount of time. So that was very, that was one really obvious benefit of pair programming that day...I did not know how to do it. I had one, there was, I guess it was the combination of the data structures underneath and some operations you could do on that and then there was a whole user interface that needed to be put on top of that. And so I had knowledge of the data structures and [my pair] had more knowledge of how the UI would look and be put together and so we had to put those together.

(T.2.13)

It also was seen to greatly reduce stress on the part of individual software developers, who no longer maintain a reliance on people who are unwilling to help them:

It's much less stressful because you can rely on other people. It's much more safe, in some kind of way. Because you aren't waiting that one day the others will help you.

(X.3.52)

I think one of the extreme programming tactics was if you asked someone to pair program you that person is not allowed to refuse. And so we both hold that very dear. You will ask someone for help they will be ready to you know, at a mutually useful time, come on board and help you out. You can't function that way. We tend, I think we tend to do that very much with each other as well as with getting the other members to do that... Yeah I think it's very important to know I can ask around because otherwise I don't think I could get, I don't think I could be effective at what I....at programming. Because I just

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

don't have all that knowledge. And I have to be able to approach someone and ask them or maybe they will know someone else who I can talk to which is fine too. And then, so we can at least put together the knowledge to fix something in the end.

(T.2.15)

Awareness of the concerns of team members was also seen to allow individuals to assess before the fact whether the other person *could*, and also if the other person *will be willing* to, engage in a discussion given their current concerns. The ability to go to other members of the team for help, to *show respect* for the time they spend helping given their current situation or workload, and the increased opportunity to be able to offer help in return, were all seen to reduce the reluctance to communicate in an agile environment. Generally, there seemed an increased ability in agile teams to see opportunities for interaction and grasp them without worrying that they are inconveniencing others. Face-to-face communication was also seen to allow for the kind of 'while you are here' interaction that cannot be instantiated in more formal communication mechanisms such as email:

But in terms of where we physically sit in the building the user experience team is separated from the project team just because we typically work on multiple projects. So it makes sense to have us sort of located in the same area of the building but not necessarily with the project team. And what will happen too is typically, you know, in the middle of the afternoon I will take a walk down to the project area as well, and as soon as people see you they're like "Oh, [Judy] I have question for you..." You know, so then they'll ask questions and things like that.

(T.5.10)

Such off the cuff interaction seems very important in improving team relations, which would otherwise involve one party dealing with a small problem in the back of their mind until it became bothersome enough that they had to go and talk to the person responsible.

I think its lot easier to get that communication going. Definitely if we didn't have the stand up meetings I know for me I would delay saying 'oh you know I was supposed to call [Tony] but I don't know...well I will call him later or I will call him tomorrow or oh he is on vacation or I am not here tomorrow. I will call him later and then you just delay things and then you end up with no feedback or feedback so late that you can't do anything about it.

(T.2.24)

This study also found interesting effects related to the dispersed knowledge of team information, in that if one person was unable to provide or communicate the information needed, there were usually other means to get around the problem at hand:

And because, the other thing is because agile is paired development, if I have difficulty talking to one developer about it, there is always the other developer who, you know, is able to communicate with me sort in the way that I understand.

(T.5.24)

Study data revealed an interesting dynamic regarding 'introverted' personality types. While some participants mentioned difficulties in involving non-communicative members in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

agile environment, others noted that such personality styles did very well in the agile environment. Introverts in the shared team environment, it seems, are given a chance to get to know others in a relaxed setting, and have more opportunity to ask questions and offer opinions. Relatively informal daily stand-ups meetings were noted as especially valuable as low pressure forums for the involvement of non-communicative team members. This could be held in comparison to environments where initiating communication with others would require entering their personal space and interrupting their work.

Overall, the ease with which communication and interaction takes place in the agile environment was seen to be very important in making the software development process much more enjoyable for team members, ensuring that communications run smoothly by reducing delay and push-back tactics, and increasing the amount of pre-emptive helping and assistance between team members.

Familiarity Through Close Collaboration

The agile team environment encourages increased communication and collaboration between team members on a daily basis. Agile team members see each other at least once a day at daily team meetings, and possibly all day in very close quarters if they are in a shared team room or pairing with another team member. One of the strong effects associated with this increased exposure was increased familiarity with others in the team. This occurred even in teams that only engaged in a 10 minute stand-up in the morning and a brief chat in the afternoon, in addition to regular project interaction.

> *I was here for about a year before we started with agile and I mean we didn't do these daily meetings and things like that and even just something like that, I have seen a big difference with that. You know, just because we are more involved and we are all sort of thrown into the same room more often, which is going to force us to talk and communicate and find out what each other's working on and stuff like that, so that is definitely a benefit...because we see each other more often, we're very, I think we're a lot more friendly with each other than we would have been before and there is always lots of, you know, joking back and forth and everyone is very, I would say it was a very tight knit group.*
>
> (T.5.16)

Agile was seen to *force* closer communication and collaboration between team members, and increases the chances that they become aware of each other. Personality clashes or differences in goals or values were seen to make themselves known at this point, whereas in a less collaborative environment they could pass unnoticed. Participants often stressed, therefore, the importance of open, friendly, and flexible people who are willing to put aside differences in order to work towards a shared goal, while personality traits such as introversion and extraversion seem to be less important. Agile team members, if they are to work together successfully, seem much more likely to have to resolve or learn to work through differences as opposed to avoiding issues or each other. There is just less room to ignore such issues:

> *I think that open and honest communication is a big part of agile culture. And that, and respecting individuals. So, um, I guess could have happened without agile,*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

certainly. But I'm not sure that it would have for us. And another thing certainly is the close collaboration that happens with agile working. Ah, a great deal of help for trust.

What were the barriers that were overcome? What do you think that it might not have happened for you if it hadn't been for that push?

For one thing I think that we wouldn't have had such a strong need for it...If there hadn't been such close collaboration, it might have been; like for example, when I first joined the company, everyone was working for himself, basically, um...it would probably just would have said, 'Okay, that didn't went well but it's not that important. We can work. And who cares? Huh?' And having to work closely together simply created the need for solving these problems. And of course, also, helped in solving the problem, because we knew each other much better. So there, there are also, a much stronger personal need for solving this problem. Because it is causing much more stress for you, and you feel much ah...<german word>. It's just no fun when you have to work with someone when you feel that. there's some tension.

(X.3.73)

If conflicts can be resolved, then the resulting team atmosphere was seen to involve much more trust, respect, caring and concern. Such strong bonds were highly related to increased willingness to both help others, and ask for help in return:

My company, that's kind of big. But in our department there are a few people that I work...it works. You actually have quite a good communication culture. Everybody is helping everybody. It is also that we have some guys from one consulting company. From a small consulting company. So they know each other really well. So when they call down and say, hey – I need this feature by tomorrow, and they can say ok. There are a few internal people and they think that they know they can call the external people and say 'Hey, I need this feature by tomorrow.' And they can do it. There are a few internal that think, ok I'm internal and I can tell the external what to do, ah, they call down...and they will say 'Ah, I don't have time. You have to do an official feature request.' <laughs>. Because, I mean, you can really understand, because they normally try to focus on their own work. To help them.

(X.4.37)

Increased interaction in the agile environment was also strongly associated with increased capacity to understand and communicate with the particular personalities in the team:

But even the daily stand ups, someone else said it, it just helps to understand how someone is speaking. You know some people and this is sort of a Deborah Tanning kind of thing you know where someone has a certain way of speaking and if you have ever met this person before you may form certain impressions about their underlying attitude. But I think definitely the daily stand up is sort of you get used to how someone communicates. And you say okay oh I guess so and so is cranky again, but he is always like that or you know he is always... And so and so was like really did some deprecating but he is always like that and that's the way he jokes or something.

(T.2.21)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *The daily contact really. It's the daily contact and just the constant communication. Yeah and I think because we had the daily contact and that constant talking, we could change the way we spoke and the way we talked to the team. You know if you first said something you are going okay well that wasn't accepted all that well. You could say okay well why wasn't it accepted all that well. Well maybe I did this, you know maybe I had too much user experience jargon in my talk or something. I will try it this way and you could kind of change the way you are talking to make it work better; like to make people interested in what you were saying.*
>
> (T.4.62)

Social events were seen to be helpful because they provide a relaxed environment where people could actually be themselves. However, there were a number of teams discussed in this study who were extremely close socially, but failed to work together well as a team. Likewise, participants mentioned teams that worked together extremely well but did not get along socially:

> *Well I think trust is so important like as I said you know the person that I didn't like personally, I respected him. And respect and trust it's all I need. I don't have to be able to go out for a drink with person to work with him, right. But I have to believe that they are doing their job to the best ability and that when we are working together, we are both being professional and we are working towards the same goal and that's all I need right. The other part is not necessary.*
>
> (T.4.70)

It seems that part of the value in agile is that it gives more leeway for relaxed communication *in the project environment.* This was related to improved working relationships that were discussed as more valuable to the team than personal relationships created outside of the working context.

Communicating face-to-face as opposed to through computer-mediated communication or documentation was seen as especially important. Part of the face-to-face environment, for example, involves an increased awareness of the state of others:

> *And that's in your voice and not only just, I think it's also in your voice. You can, if you really are listening you can hear how people are doing in their voice ... I can hear your voice, are you tired; tired is a big one. I can hear people when they are tired. They become flat. And so you don't know if they are tired or in developers case whether they are fried or burned out or are they stuck on a problem they can't express it. Developers are very scared to show they don't know the answer. And so you have to get over that.*
>
> (X.2.6/9)

This perception of state of others was a seen as essential to team relations in that it colors interpretations of their actions. Awareness that someone tired, burned out, or stuck on a problem, will likely impact how their actions are perceived. Thus face-to-face awareness was seen to smooth team relations by increasing understanding of others. The familiarity that comes from close collaboration in the agile environment was seen to allow a better understanding of a person over time, and their stable characteristics that need to be dealt with in working interactions:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I guess with all projects if you have someone whose personality is different than yours, you have to recognize it. Like you have to recognize okay the tension is coming from a personality issue right. They like to work this way. I like to work that way. Neither of us are going to change. We just have to make it work one way or the other not let that affect what we are trying to achieve basically.

(T.4.41)

Such familiarity was seen to open up avenues for open communication so team members could bring up real problems and issues in a safe environment:

Well, just that because I see them so much more often, and they see me; because we don't sit together, you know. In the beginning when we working on project, and even for a while before we were agile, you would see someone and go over and say 'Do you need any, am I blocking you on anything?' And they would say 'No.' You know, because they don't really know you well though so they wouldn't have a lot...there wouldn't be a lot more discussion other than 'No, I don't need anything from you right now', or, 'Yes, I need such and such a design.' But now that we see each other everyday, and a few times a day it's like 'Oh, how was your weekend?' And, or coming out and saying, 'Well, I don't think this design isn't going to work because such and such.' You know, you are less afraid to speak your mind because you have more of a relationship with people that you work with.

(T.5.26)

Another important effect is reduction of the amount of *mis*communication in the software development environment. Miscommunications were a common point of discussion in this study, and came from the fact that people are talking past each other due to a lack of common understanding.

The technology persons would go into their boss and say 'We are dying down there. We are just working so much overtime, we got to have one more person, we just have to or we are going to kill ourselves ' and the manager sitting there thinking 'Oh I don't understand the problem - You are getting the work done so what if you are working overtime that's what all software people do. You are getting the job done you know so what? What's the deal here? I am getting a lot of work for nothing you know I, I am paying you salary.' So I go into them and I say, 'Here look let's reframe this. Let's talk about how much you can...you get one more person, how much you can improve productivity and how that's going to hit the bottom-line?' So they got into their boss and they say 'Look, you know we had one more person down there, we could improve our productivity like this. And we will get to market earlier and therefore we have a better shot at this much profit.' Overnight they got their person because this guy gets it. He gets it.

(X.2.16)

Understanding of others in the agile environment, and a resulting shift in communication tactic, seems to increase the chances of both parties being satisfied in getting what they want.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Increased Dialog and Discussion

Much of the value coming out of cohesive agile teams was related in this study to increased dialog and discussion. Some participants expressed great enthusiasm regarding the task of making a case for your own ideas and generating buy-in from others in the team; other participants viewed it as somewhat draining, but accepted the value that it brought to team interactions.

Equal partnership and good personal relationships seemed to be invaluable to increased dialog and discussion in the team environment. Cohesive team members showed a willingness to question the motivations of others rather than defer to those in positions of power, or to avoid discussion of issues with team members because they were unsure of how they would react. Collaborative as opposed to command and control relationships were related to the ability and willingness of team members to ask questions like, 'Why are we/you doing this?' and expect an answer. Thus an environment of equal partnership was seen to increase the need for all parties to make their point by communicating their underlying motivations to the team:

> ***What was the purpose of these meetings communicating the usability test results?***
>
> *Well it was to make sure that they understood why things had to change, why that some things didn't have to change. If it was working really well to give them positive feedback on stuff that was working really great but also so they would understand that when we made a request for a change why we were making that request. That we weren't just saying well today I don't think this should be this way or something...that we had some good reasons behind it which is why we would kind of focus on the big stuff.*
>
> (T.4.16)

This seems to be extremely powerful, and related to the fact that agile team members were much more aware of the reasons why they did things in the team environment, in addition to exhibiting an increased awareness of the opinions and motivations of others. It seems that agile encourages team members and team leaders to *really think* about what they are doing; to consider how their needs aligns with both team goals and the personal tasks and interests of others in order to justify their actions to the rest of the team, and convince others that what they are doing is of value to the collective endeavor.

It follows that if team members have to sell their ideas to the others in the team and provide a compelling argument towards this goal, that this will strongly effect motivation and buy-in, both on the parts of the people doing the convincing, and those being convinced; particularly if team members have worked to frame their needs in ways that other members of the team will understand.

Engaging Communication

A common comment in this study was the fact that developers are often bogged down in technical details and fail to realize that other people don't find these details as interesting. Increased awareness of others in the team environment, and increased understanding of the personal differences increases the chances that team members can and will cut down communication to only the necessary details as indicated by the person receiving the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

information. In addition, they are encouraged to make the communication engaging and compelling in order to produce agreement and excitement in other in the team.

Having an interested audience was also seen to have a strong effect on team communications:

> *And you know you get this positive feedback from what you have said, and so you would want to keep talking about it. Like when the marketing guy came over and said 'Okay well I am working on this deal,' and we were like 'Oh tell me about the deal! We think this is really cool.' You know he is going to come back and talk a bit more about the deal and stuff he is working on because he is getting a positive reaction to it. As opposed to ' Oh we don't care about what that is. Tell us when you finally seal the deal.' You know, no one would ever say something like that. So it was a very positive atmosphere on the team.*
>
> (T.5.4)

A common comparison made by participants was between the usual dry software documentation and such team member interactions in the agile environment. In this situation the presence of an interested audience works to increase team communication and motivation.

This situation also illustrates the value of unfolding developments in the software development project environment. In the agile environment, understanding and interest are generated and built upon as time goes on through frequent interaction and progression of events. This can be held in comparison to a situation where one role/group creates a large document containing all of their understanding, which is then passed off to another role/group who must digest the everything at once and without the benefit of any contextual information. The value of *storytelling* in the team environment was a common theme coming from discussions with participants.

> *Telling stories. It is very powerful, because I'll just tell people stories. 'Well one time this happened to me' or 'one time I heard about this happening to somebody' or 'One time this happened to somebody.' Famous one very early on in the days was when I taught computer classes. Students come in just totally fried with a print out this thick for a one page program and I said 'what happened!?' 'Well I don't know just started printing and printing and printing' ...'I don't know' I said 'maybe you got a little infinite loop going around. Hey - everybody it happens to them.' And maybe I'll tell him a story about how I wrote a piece of code and I wrote a infinite loop to put a 0 in order to clean it out to put 0 somewhere and then I just I wrote a piece of line of code that said put 0s here and it went in infinite loop incrementing zeroes and I will bet pretty soon it stopped. So I tell him these stories and they go 'Oh so she makes mistakes too, so I'm not so stupid.'*
>
> *(X.2.25)*

It is not entirely clear the value of stories in the software development context, but it seems related to both increasing understanding between team members, increasing understanding of the different parts of the collective team endeavor, and in stimulating thought and action in a non-confrontational manner. Traditional means of communication such as documentation, on the other hand strip work-related communications down to essential information. The agile

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

environment allows team members to share experiences and problems surrounding their work as they engage closely with others on a day-to-day basis. Such interactions provide rich contextual information, anecdotes, reasoning, and underlying motivations for the actions taken and decisions made. Evidence presented in this manner seems to be much more compelling and motivating than a dry document presented free of personal meaning.

Checking Behaviors

Constant interaction and familiarity with other team members seems was seen to be highly related to checking - a common form of behavior that was strongly associated with cohesive teams in this study. Basically checking behaviors consist of asking the other person if they are okay. This was seen by many participants as essential conversations that occur at social events or in face-to-face environments:

> *I always ask these questions that go, 'Are you okay? Are we okay? Are you mad at me or is everything fine?' Not just, you know 'Is everything fine?,' because I will say 'Oh yeah'. But you just say 'Are you mad at me?' 'Well, no, why would you think that?' and it becomes a discussion*
>
> (X.2.6)

The familiarity fostered in the agile team environment was seen to supports checking behaviors, as it allows people to sense that something is wrong with a person and ask about it in an appropriate manner, and also allows for a more honest response in return. Checking behaviors in this study often occurred with regards project concerns, but were much more likely to be associated with personal matters. Such behaviors, the concern they demonstrated, and the communications that followed, were strongly associated with feelings of closeness and security in team relations:

> *Also, it has done a lot for our communication style. So that, recently...I don't remember what it was about that...we had some discussion about, it must have been some minor thing. So we just discussed and we settled the issue, and when it was lunch time before the coworker was just on his way out to lunch. And turned, and came to me and said, 'What I said in that discussion. It wasn't meant that way. You might have understood it in that and that way.' And I was 'Okay, no it was all good, I didn't misunderstood, but thank you for telling me.' And...we had, it was a great feeling of trust between us, because we could speak openly with each other.*
>
> (X.3.71)

Checking behaviors, when combined with an awareness of others and their personal goals, were seen to greatly increase the chances that the team environment would support individual enjoyment and development in the team environment:

> *And then because there is an [internal team member], which is myself or somebody else around, we'll also sort of go 'Hey are you driving more like you said you would?' You know, just something. Not, you know, standing over peoples heads but just like – 'Have you shown each other?'. But I mean it's also up to everyone to sort of make sure it's like everything with communication you need to make sure that the people know, you know. And some people would be reluctant to talk about it ... but then if they're*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

reluctant then you will do more of the – 'It would be really good if you...' 'Are you driving more like you said you would?' Something like that, you know, just like little things that your mentor will try to make sure that you do.

(L.4.33)

The interesting thing was the value of both the informality and the frequency of this kind of behavior in increasing security of individuals in the team environment, and in gently nudging them towards actions and choices such as accepting team practices, buying in to team goals, and pushing themselves to excel in the team. This behavior, in addition to ensuring team members that everything on the project was running smoothly and that there are no glitches in communication, also seemed to be a means of constantly reaffirming relationships and the team identity.

The nature of the agile environment - which supports openness and honesty, which supports helping others, and which supports continuous awareness of and communication with others - greatly increases the occurrence and the effectiveness of checking behaviors in the team environment. Again, this can be compared to a traditional software development environment where individuals are separated by cubicles and are much less aware when others encounter problems, and usually less enabled to ask about these problems or help resolve them.

Being Wrong

A very common theme in participant discussion of the team environment was the importance of making mistakes and additionally admitting mistakes once they occurred. While many people are unwilling to admit being wrong, it seems that this denial of fallibility is especially prevalent in the software development community:

They were new to pairing too. And that was kind of interesting, because they were really nervous about appearing silly in front of other people. And so they didn't want to say anything until they had all their chickens in a line and until they had thought the thing out totally.

(L.3.7)

And I would just stand there and say how is it going? They would say 'Ah.....' and they wouldn't, and the other one is they wouldn't look me...and this one guy wouldn't look me in the eye. They won't look you in the eye when they have got a problem they can't solve because...they are the pros and they are supposed to be able to fix everything.

(X.2.29)

The common thing is that people have difficulty because they don't... it takes them some time to accept that they are not going to know everything. And once they accept that, then it's, you know, then everyone is, I think it's quite difficult for somebody to come into a company where they don't feel like all the answers will be somewhere and there won't be any exceptions and they will know everything.

(L.4.21)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Software developers who are unwilling to admit they are wrong are fighting a loosing battle, it seems, in a software development environment that includes steadily increasing complexity and new technology. Knowing everything is practically impossible, and there is an increased need for members of software development projects to heavily rely on each other:

> ***Could you describe what you think it is about the team that allows you to get along so well together?***
>
> *I think because it's sort of an open and friendly environment, and I think partly because, well I mean it's open and friendly and also the fact that. Well it takes some getting used to, but when you join a place you don't feel on top of everything because as I said there is so much different code and so many different systems and so many different languages and you are only going to have a level of expertise on some of them. So, you know, you have to rely on each other because you have to accept that you will never feel, not never, but it will take a long time to feel on top of things, totally.*
>
> (L.4.19-20)

Being willing to admit mistakes in the team environment was shown to be highly associated with improved personal relations, and increased helping behaviors. In one case, for example, a participant had suggested a course of action and her expectations were exceeded when certain others admitted that they were wrong and were responsive to her suggestions:

> *And next meeting, I come and there, and they have done it! I was so thrilled that they did it, [and] she calls me now and again to come work with her organization and do stuff. I will go in a minute. Doesn't have to ask twice. Because she took the courage to say we made a mistake, and we will fix it. And when they said it nobody in the room could look at me...they didn't look at me and I went – 'Okay I got it,' doesn't matter how I get there, it was so cute. But we got that to happen.*
>
> (X.2.111)

This study found that unhappy and non-cohesive teams were strongly correlated with environments where team members were unable or unwilling to admit fallibility. This was seen to result from constraints in organizational/industry culture, lack of personal contact with others, and/or personal reluctance on the part of the workers themselves. Participants in cohesive teams, however, tended to comment on how the environment or certain situations exposed underlying motivations and even vulnerabilities on other people in the team environment. This in turn lead to a deeper understanding, increased empathy, and a corresponding willingness to communicate and collaborate in an open, respectful, and trusting manner.

Sharing and Support

Increased exposure in the agile environment was often seen to result in stronger social bonds:

> *We go out for everyone's birthday and things like that and...so, there would always just be lots of discussion about, you know, who was leaving and why and, you know, everybody wants to keep in touch, so they are all sending out their e-mail so, you know, it's really, it really made an impact. Where I don't think there was, I think there, well there might have been some of that before but it's definitely, significantly more now.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For quite a few people there is definitely a more social relationship now just because of that increased exposure to each other.

(T.5.18)

The relatively relaxed social environment associated with agile teams and increased exposure to others increases the opportunity to share ideas, opinions, progress, and concerns. Agile environments were also seen as generally more likely to be playful, sociable, and less formal than traditional software development environments. The interaction and social support provided by the agile environment was highly valued by many team members:

And there are some days where I just feel like I need to talk to somebody because it gets kind of boring if you are just stuck with your own bugs and sure you can keep working on your own bugs and plugging away at that but there are other people who need a second pair of eyes for their code as well. So, we do have, we try to code review each other's codes all the time but that's often offline...once it's been submitted. And it's not as interactive.

(T.2.10)

Awareness of the opinions of others also seemed essential to a sense of personal security and control, and checking in with others was seen as a way validating, enhancing, and refining ideas.

Do you find that there is a difference between having that person there in the customer role and not?

Ah, well it's mostly because there isn't somebody to clarify the work. So now I'm just trying to get someone else and ask their opinion of it. Whereas if you have them right there you have a better idea of what the customer actually needed

(L.1.8)

And even before I started agile and pairing full time I found that If I had any kind of design idea, then I would find someone else and just run it by them. And it didn't matter if they had any kind of experience or not. Just the act of explaining it to someone, and them asking questions about the bits they didn't understand about the explanation. Then it makes everything make much more sense.

(L.3.30)

Such interactions were particularly valuable in ensuring that the individual efforts are truly contributing to business and team goals. Constant and shared dialog were noted as resulting in well thought out, elegant, and mutually agreeable solutions; more so than if one person had come up with a solution on their own.

Another aspect of the agile team environment was the ability to share small successes. This occurs on a day-to-day basis as individuals overcome small hurdles, often with the help of others.

It's no fun celebrating alone. And so giving someone else a high five. That is a rewarding experience.

(X.3.52)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The ability to sharing problems with others on a day-to-day basis seemed to increase the amount of social support and guidance team members received; this was often noted as a surprisingly good experience for individuals who were more used to working in isolation. Increased familiarity and presence in the team environment also allowed for inspiration resulting from the enthusiasm and encouragement of others:

> *I think another part of it is getting inspired by others. The keynote yesterday morning, is very important I think. Inspiration you get from...enthusiasm from each other, and inspired by each other...you can't do that alone <laughs>*
>
> (X.3.55)

It should be noted that although all participants in the study spoke to the benefits of close interaction with peers, a number of them noted that interactions can sometimes be draining when not managed properly:

> ***Do you find pair programming draining at all?***
>
> *Yes. It's very intense and very draining. When you are by yourself you can kind of sit there, and you are working away, and you can kind of drift off there from time to time. And your mind kind of needs those rests. You don't get those when you are pair programming. Because your pair picks up, like if I'm about to call a method and I forget the name of the method that I'm about to call so I pause ... then my pair will have known what we were going to do because we have been talking all the time, and will prompt me on it. So you don't pause even for a fraction. As soon as you run out of ideas on how to solve a problem, then you stop typing and your pair will grab the keyboard and say – 'Well what about this?' And you will immediately keep on going at the same pace instead of having a break while you stop and think about it. So it's very intense, yeah.*
>
> (L.3.12)

Mentoring and Learning

Finally, one of the most important aspects of agile software development to participants in this study was the ubiquitous teaching and learning that occurred throughout team interactions.

> *The other thing is that, ah, there are a lot of things...it's a much better learning experience. You continually learn something from the other people.*
>
> (X.3.52)

Team members had the opportunity to constantly observe and learn from others as they went about their day-to-day work. The major means for mentoring in this way occurred through the agile practice of pairing. Pairing would usually occur between developers in the form of pair programming, but would also involve different roles collaborating and helping each other get up to speed in their respective domains. Another way that the agile environment was shown to support learning was in the estimation and story selection processes, which better allows team members to work to suit their own abilities and pace:

> *What we ended up was that the more experienced people would tackle the hard ones, and the less experienced people would settle the easy ones ... they would take those*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ones that were known issues. So that kind of worked out as an easy way of easing into it. We were doing paring as well. Pretty much of 100% of the time we were doing pairing.

(L.3.6)

The learning environment was highly associated with increased communication in the agile environment due to knowledge overlap:

There is another team that we sit beside and they are so quiet you don't hear a peep out of them. You don't really know what - I think each person very much has their individual specialty; whereas in the team I am on we try to share the knowledge a bit more between programmers. And in that way I consider us to be more agile, or adopting more Agile development things. And I like that.

(T.2.3)

The personal value of learning is, of course, a strong source of enjoyment and motivation. One finding in this study, however, was that seeing the development of *others* was one of the most important sources of enjoyment and inspiration in the team environment.

And it's a lot of fun. They are very amazing people. They even went to go to meeting and they went on, they had a lot of research and how they can get some online training and they have to go visit people and so to get more response time to them. And just amazing to watch them. It's really fun, really happens. So I get very excited about it, so I really see it worked, yeah.

(X.2.70)

And that eureka moment is really quite nice actually to see but you almost have to say be patient you know.

(L.4.27)

I get most of my joy from seeing other people...seeing them improve, you know. Seeing them get better. And the process, and their work, and the way they are developing. It's great

(O.2.4)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Individual in the Project Environment

Project Goals

A Clear Objective

> ***Have you been on a really dysfunctional team?***
>
> *Um. In some ways that spec driven team was a bit dysfunctional, but that might have just been my perceptions. Um...it's bits where there's more push to meet the spec than to meet the customer needs. I mean, in my view that's dysfunctional, but in terms of most software engineering teams, it's not.*
>
> (L.1.82)

An extremely valuable aspect of agile software development methodologies was seen to be that they help teams maintain focus on a singular overriding goal – to deliver the most business value to the customer in the shortest amount of time. This goal is instantiated in a rigorous process of identifying business value in the form of stories, estimating time to develop these stories, and making choices for inclusion in each iteration of software development. Such a goal was seen as valuable not only from a business perspective, but from an individual perspective. Delivering business value is something that all participants in this study saw value in, and which all team members were seen to agree on, and they were seen to appreciate the fact that bulk of effort throughout the day in agile methodologies is spent directly contributing to this goal.

Continuous Integration and Unit Testing

Continuous integration and the presence of a working version of the software was shown to be very important in the cohesive team environments. An interesting effect found in the study was the importance placed on the presence of *working* software. It seemed to provide a sense of security and comfort under which team members could work with reduced stress and worry:

> ***Interviewer: could you describe what happens in the team room when the traffic light turns red?***
>
> *I can't say more than there is some kind of tension. I think that everyone is just aware of the fact that, um, when the build is broken, then there is something now that....ah, the other way round – when the build is succeeding, you know that when the build is working, if someone needs to use it, then someone can get the latest version and use it. And, um, when the build is broke, you don't know when the next time will be that what you are currently working on will actually be useful, to the customer, to the product manager. You can tell, um, that what you have done is what I wanted...what is very visible, is that once the build succeeds and then there is much relief. It's is... '<Phew!>Cool.' And we can work safe again. Most of our time it is green again.*
>
> (X.3. 8-10)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The *visibility* or tangible awareness of whether software is working or not working was also seen to have strong effects on agile teams and their motivation to fix a broken build. Another example was a story of a team who had a lava lamp as an indicator that the build was broken. One unforeseen effect was that because it takes 20 minutes for the lava lamp to heat up and start bubbling, there would often be a mad scramble to fix the bug before the lava lamp heated up. A working software build also seemed to be kind of a benchmark against which team communication and collaboration could occur. It ensures that everyone is on the same page, and was seen to have a positive effect on increasing awareness and understanding across the team of what needed to be done to complete a task:

> *But I also install the latest version of the software everyday and I can kind of just see. I mean I'm also, I get like the bug, e-mail bugs and things like that so I am able to sort of identify what, you know, what the current problems are and things like that and, you know, if there is anything I need to address and stuff. But typically too before the stand-up meetings at least one person will have installed the daily version of the software and they'll kind of just say, you know, this is the state.*
>
> *(T.5.21)*

Having a working version of the software was shown to reduce and ease communication surrounding the finished product, and to increase the ability for team members to ensure their requirements were met in the software, and that they would be happy with the finished product. This seemed especially important when it came to user interface design:

> ***Do you find that it helps you to have that working version of the software?***
>
> *Oh absolutely. Absolutely. What I do is, well, and I don't mean it as negatively as it might sound later when you are listening to it, but I go in and I look to make sure that my stuff's been implemented correctly. So, I do that and then it's also just, you know, I will go through it and I will look at the stuff I have designed and how it's been implemented but also just play around with it to make sure that it feels, you know, the way I expect it to feel. Like does it, you know, in terms of the interactions does it feel, is there an action that's unexpected, is essentially what I am thinking about; you know, does the design work the way I thought it would.*
>
> (T.5.22)

Miscommunications, therefore, became very visible early on in the process, and allowed people to better understand what was *not* being communicated from their side of the equation, and that they needed to work harder to communicate their point.

Continuous integration was also seen to establish a high level of discipline and rigor regarding team conduct and software development, and to simultaneously increase team cohesion:

> *The team pays a lot of attention on quality so one of the things that Agile is introduced here, for example, it's been automated testing, so the team has just gone nuts I think. It's probably the best way of putting it on automated test and we have no other product in the building like it. You know basically we have been...the team has been shipping [the product] now for about two years. And on average there has been no more*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

than about 50 outstanding bugs on the product. And so it's been pretty constant overtime and I think the team is justifiably pretty proud of that. And I think this really brought them together

(T.1.8)

Continuous integration was also seen to ensure that developer effort was relatively well focused on contributing to the working software system, in that w individual is not successfully integrated into the system, then the team knows straight away and can work to resolve the problem:

There was another guy who was again kind of inherited. Who was ok, he was a bit...he knew the domain very well but he was a bit careless. Even after a year he would just check stuff in without checking that everything ran and everything. And then we would have to back it out and go around the loop again a couple of times.

(L.2.5)

Another factor associated with continuous integration was that it reduced the need for any communication or dispute surrounding around what constituted task completion. Continuous integration in the agile environment makes it very clear that a story is completed only when it is fully integrated into the final product and all the tests a working. In addition, the goal of working software is a clear goal that individuals and teams can work towards.

It's worth noting at this point that the majority of developers, when asked the agile practice they value the most, would respond that unit testing made their job much easier:

It definitely, well I think, agile or not, I think automated testing incredibly improves, well I guess I'd say 'programmer happiness' as well as product stability. ... it definitely lets us be more experimental with making changes and developing features. So if we develop a new feature or even if you are sort of adding on to an existing feature, it's easier to explore different ways of implementing something. So I can try going down one path of how I think the code should develop. And but then maybe I don't want to go that way, maybe we want to change direction go another way and it gives us lot more confidence to try that.

(T.2.49)

Unit testing was seen as a means for developers to ensure quality and take pride in their work *in the context of the whole team*:

A couple of people in my team that are personally affronted that when somebody finds a bug in their code. And I think that's a great thing. I love to see that more, I don't...you know I have been in past jobs you know where you know manager will go storming in to somebody's office because such and such a bug was found at a customers site and you know my God you know, it's the end of the world kind of thing. And I would rather have that coming from the developers.

(T.1.33)

Automated testing was also related to the ability to coordinate development efforts across people and periods of development, and to feelings of control over team development efforts:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Well for them it's more, keeping the automated tests up-to-date and they do things like somebody reports a bug, and they will put a fix in when they put a fix in they will also put a test in to make sure that that bug never recurs again. And that's very different from the other teams where the same bug you know gets fixed and then crop up again a year later because somebody changed the same code and that kind of thing...And so there is a real attitude difference you know that I detect in the [agile] teams because you know they do bug fixing but they do it constantly and for shorter stretches. And so they are always on the top of it and they never have this huge backlog that's you know staring them in the face and making them feel like they are not in control.

(T.1.28)

Constant Awareness of Project Status

A very clear awareness of the status of an agile software project is created through daily stand-ups, information radiators, pair programming, an integrated code base, and presence in a shared team room.

...So you know, check who is working on each task everyday and ask them to estimate how much is left from what they are working on so that for each task from day to day you could see whether the stories were going to get done or not. You know so it was sort of as transparent as possible. And you know the project manager was very supportive and didn't sort of beat the team up when it looked like it was - well, not much - when it looked like we weren't going to make it. But yeah I guess he did a little bit but you know, generally he was more in favor of honesty rather than just being told that everything was going to be alright.

(L.5.22)

The alignment of visible or official project status with the *actual* project status was an important factor to team members in a software project:

What was the difference between in this environment and the [cohesive team] environment?

So there is a guy in this project here who is the project manager. And he was very slick and he knew what he was doing, and he was realistic. And you could talk to him. So whatever was going on in our division...we were still under pressure to deliver, but it was realistic. And it didn't' involve mass panic and unrealistic deadlines to deliver and the whole team staying late. And it didn't involve dropping quality...so a lot of the problems were trust and plausibility.

(L.2.15)

This alignment was also related to the reduction misunderstandings and communication in the agile project environment, and increased ability to collaborate as a team:

But there is a lot of these big organizations that have this myth of - 'well that's not how we build software.' But when I ask them how they do build software it's just this chaotic blur of noise that comes at me and you can see them...you can see two people say two different things and look at each other and say 'What?' And you know, they may

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

have worked together for three or four years because they don't have a holistic understanding of how they build things. And I mean, that's one of the biggest differences.
(X.1.4)

Agile team environments support the transparency, awareness, and realism in the project environment that is required to allow people to work together in a cohesive manner. Agile was also seen to provide information on a much more detailed level than in traditional software environments. An absence of this detailed project information was related to feelings of discomfort and lack of control:

I felt uncomfortable at the moment that I was further away from where things that were happening. And happening in the sense that things were happening in the project and I was really in one end of the hallway, three persons without a connecting door, and you just didn't hear what was going on. And maybe it was just my, ah, I like to hear when things are going on <laughs>. But yeah, you heard less about things already being solved, or why the build was not running at that moment. I felt less comfortable as well, and it also had a slight, I mean it's not really measurable, effect on the quality of some stuff. Because things were just – it took a bit longer for things to be communicated. ... it was a real difference from the old location where you were cramped into three big rooms, which were basically located on a very small hallway, so when people were yelling something in one room, you were definitely able to hear it in the other room, and that was gone.

(X.4.20)

As mentioned in the section on team awareness, study participants greatly appreciated, and were strongly motivated by, the fact that they could keep their finger on the pulse of an agile project. The importance of this information to participants existed separately from any kind of individual role in the issues being faced by other team members. Agile team members liked to be up to date on a daily, or at least weekly basis as to the status of the project – good or bad. It was important to them to know who was having problems, what the problems entailed, and what was being done to fix them. One participant describes project slippage in a 'waterfall' based project.

But once that starts slipping, and especially what is happening with this other team. That it slips, but it slips in some kind of uncontrolled way. Because they say, right before release, 'hmmm, hmmm, hmmm, hmmm, maybe we make it...but, but....ah no we don't. Maybe tomorrow.' But this maybe tomorrow goes on for a whole week. And that's, ah, you are not in control when that happens. You don't know where you are standing. You need to analyze of course if it's really a delay or if it's not stable enough. So it's not clear, what the reason is. And that makes me very uncomfortable.

(X.4.12)

This was a common comment about projects in which schedules would slip and there would be a lack of awareness about *how* they were slipping, or *why* they were slipping, or *what could be done* to stop the slippage. Having up to date information on project status presented either at a daily stand-up or visible in an information radiator was seen to allow team members to be aware of what was occurring in the team, and thus be able to help others in the team in working towards project goals.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The 'big picture'

Seeing the Big Picture

> *So leaders basically ask a lot of questions in my mind. They force the team to think clearly. Developers are so close to solving the problems of software.' It's interesting that agile seems to give people a bit of tunnel vision in that they sometimes get stuck in iteration considerations and stop looking at the big picture.*
>
> (X.2.13)

Programming requires a lot of detailed focus on the task at hand. The agile environment seems valuable in that it encourages developers to look at the big picture as well. In discussing information provided in daily stand-ups, one participant mentions:

> *I find it definitely gives me a better idea of the product as a whole. So I can better understand, in the end I can understand who the product is aimed at and what that market is and who is in that market and I think its more of a bigger picture about what problem the product is supposed to solve.*
>
> (T.2.28)

A big picture or holistic understanding was highly related to team cohesion in this study in that it helped align goals and encourage people to work together rather than focus on completing individual tasks. Holistic understanding provided motivation for many team members in that they could see and understand where their tasks fit into the larger endeavor, and it helped them maintain focus on larger team goals:

> *You will have this thing with the completed stories...there are the in play stories, and there are the ready to play stories. So you are able to see, you get an idea of what work is coming up, which is very good. Because you get an idea of, you kind of know where it is all supposed to be going. Doing just this next story, and not really understanding how it contributes to the big picture leaves you sort of short of context.*
>
> ***How does having that context make things easier?***
>
> *Um. I don't really know. Maybe other people don't look at it this way, but if I know how the whole thing is supposed to be fitting together, then, it's kind of like trying to do a jigsaw puzzle without knowing what the whole thing is trying to look like.*
>
> (L.3.21)
>
> *Seeing the big picture really allows you to know what you're delivering each month, what everyone is delivering each month. And that just makes you know where you're stuff fits in with everybody else's.*
>
> (T.3.21)
>
> *It depends very much on the personality of the developer I think. Because some people don't care, they just want to get on with their task. But I think context helps a lot because you are sort of trying to think – what are we doing this for? And that's where acceptance criteria also kicks in. Like just to remind you, what the hell are you doing*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this card for, and you know and it might be a great piece of code but that's not what they asked for and that sort of thing.

(L.4.58)

There was definite relationship in this study between having an idea of the big picture and the amount of motivation, purpose, and excitement regarding work in the project environment:

So in the stand up meetings I find it more useful to start asking our management, our director you know are we going to be, you know when is this contract with this particular company going to be signed? You know. Are we, have we been, did we sell them a seat? You know, were you successful? You know what is this trip for, and why are you talking with these people and when can we go?

(T.2.28)

The agile software development environment provides more context surrounding development tasks - in terms of business value, in terms of what other members and roles in the team are doing, and in terms of how everything fits together in terms of working software. This was seen to allow for increased discussion in the team environment surrounding solutions that would best satisfy all needs, rather than one party specifying their needs and others trying to fit around them:

You know I am trying to think back in the old way when I was on [another larger project]. I think those user issues were very much filtered. So by the time it got to me, it was very much more explicit how it was going, what it was going to do and how it was going to behave; and not a lot of background as to what the original task was and what the original situation was, [which] makes it harder to question ... Whereas I think definitely on this product there is a lot more interaction that way and we can go back and forth and say 'You know I know a customer said he wanted this but I think really it would work better if we gave him this,' which to me makes a better product definitely. I really got the sense of that being a problem in [on the old team]; where you work, you work and work and work on a feature, in the end it isn't exactly what they want but at that point it's too late and I think in agile you at least modify it on the way and get, end up with a better solution. Does that motivate me better? Yeah I think so.

(T.2.48)

Participants who would talk about their work in terms of the big picture tended to be more animated and interested in what was occurring in both their project, and in their organization. These participants were also more likely to say things that exhibited initiative towards improvement. This could perhaps be related to an increased feeling of freedom to act due to possession of a more holistic understanding of situations and events:

Very important is that we have all the company information up on this wall right here. And we have our top ten customers, and what the plan was, and what actually happened. Which, well this year is <grumbles>. Not so good. Hmm. But it's good to have it up there. It's good because they know that we are not bullshitting them. If it's not going well, well at least you can see why. You can see what products are creating revenue, and what is happening; that things just changed because the market moved. I think that's it's

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

good that we show this. It's really very good. Yeah. It's fair. It makes them feel, I think, what I would like is to get people who feel like entrepreneurs. Which is hard when you are dealing with people who are employed because they don't naturally work this way. Because they are working for someone. But if you are a little more, that means that you are more attached to something and you care more for it. Meaning that you can only be an entrepreneur when you know all sides. Not only the product, not only the customer side, but the business side. I think it's important, you need this to be involved; to understand all sides.

(O.2.11)

A big picture understanding was also associated by participants to better design, improved ability to integrate their work into the software product as a whole, and generally more robust code that could survive future changes:

But yeah I think [a long term vision] is important to me. It is very important to know where something is happening because you end up with better designs. ... knowing where things were heading affected our initial design decision and which is great. Because now it's survived, it lived and it's been robust enough that we can do all these things but because we had an idea of where people were heading. And so now they are saying well maybe we don't want it to be a file on disk, maybe we want it to be accessing data from a database and then we kind of got a whiff of that before... And that is true business value because when it comes to a point where we have to do somebody's crazier things you know we can say yeah we can do that, we are using time to do it; which is fine. But once you tell me that's the priority and that's what we are working on then yeah, it's doable. I don't have to go back and rewrite everything or rip everything out. Rewrite it. It ends up being much more palatable.

(T.2.33)

Related to the idea of having a big picture is the effect of the presence of a final end-goal above and beyond the weekly iteration. Several participants mentioned the *lack* of a feeling of 'buzz' or momentum in the team environment related not having a definable finishing point for the project. For example, one participant was involved in product development:

Well this is good when it works. You work out what you want to do and then you go and do it. It is hard not to have an external goal, though. You don't have someone with an idea of what's going to be happening. I mean, I don't have any idea of what I will be doing in two weeks. I'll have finished all my post-it notes. And then I'll be on to something new. I mean it's kind of exciting. One of the difficulties is the fact that it's never going to end. Until it hits the point were adding new features costs more money than the sales of those features. And then someone will have to make the decisions to stop. I do find it a bit frustrating that we only have one product line. So you can never have a break from the software. In the previous job I worked on maybe five or six different projects through the course of the time. So there was a sense of closure on the project. So it was, ah, evaluated, and you can just forget about it and never think about it again.

(L.1.31-32)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I think you know, teams that have a sort of...it's not very pro agile thing to say but I think probably one of the differences is where projects where there is a definite deadline, and a definite definite goal of some project that has to be met. A little bit more, you know, just looking a bit more focused when there is some sort of fixed deadline and some definite stuff that needs to be done.

So you find that more motivating?

Yeah, whereas there are some agile projects where there is a burn rate, and you are sort of, you are doing as much as you can every iteration, and you know you are doing the highest priority stuff. But you never, you know, you never,...I mean it sort of is what agile is all about, but you know there is no point at which you really sort of finish. You know, you're always adding new features and always doing more stuff. So it's not – there isn't really a sense of completion that you get with a, effectively a less agile thing. I mean I've never really thought of it before, but I guess that's a little bit true. Yes I am going to have to think about some way of getting around that.

(L.5.18)

This was related to a very common theme in this study, which was the sense of completion. Participants related a definite preference for seeing that they have finished something so that they can put it behind them, and an aversion to having partially completed tasks hanging over their heads.

Exploring Information Radiators

Agile teams in this study made varying use of information radiators. Such artifacts were found to be extremely valuable, however, in showing the 'big picture' state of the project, and increasing motivation and engagement in the project environment:

You mentioned that the team whiteboard helps you see the big picture. Does having that big picture change how you feel about the work you are doing at the moment?

Yeah. It's fabulous. I was one of the people that encouraged that we had a big whiteboard, months ago, I am not sure even how long ago it was now. Because in the past it was very much, on other projects or teams that I was on, it was very much you define a bunch of features and then you worked on them for months and then you bug fix for twice as long and that was the release. So you really didn't know what the big picture was, ever. You just knew that one of these or a couple of these features were what you were supposed to work on for however long and then you are just fixing bugs for ten months or – well that's an exaggeration, but that's pretty much what it felt like.

(T.3.18)

The main use of information radiators found in this study was in giving agile team members a better understanding of the project as a whole. The general use of information radiators was to display the stories for the current iteration in a way that was clearly visible to all members of the team. This increased the ability of team members easily see what was going on

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in the entire project – the context, the dependencies, changes in status, and the tasks assigned to team members for the current iteration:

> *And the purpose of it is so that, there is one spot where anybody on the team can see what's being worked on, what's about to be worked on, you know, what's coming up in the next release and release after that. It's convenient in that there is the one spot, you know. If you are a manager you can see the whole big picture of the project and you can tell that you've got something here and something over here and it puts all the tasks in context in one glance, so you can look at dependencies and if you move something from one part of the board to another part of board, you know right away well something from that part of the board has got to be shifted down, you can see the ripple effect.*
>
> (T.3.3)

> *And then also so that when project managers walking past or even us you know, just interested developers are going to walk past and say okay I can see how this is going: someone is working on this because this is a place holder; this is like ready to demo;this is ready for signoff. You know, that sort of visual cue. And also they have tried to separate the cards into the following iteration. So that you have, if you walked around the room, you should have a sense of what's coming up, what are we aiming for, where are we now, you know this sort of thing.*
>
> (L.4.55)

As noted in previous sections, this awareness of team activity was strongly related to a sense of security and control.

Information radiators were seen to be exceptionally valuable in providing visible indicators of progress on the project, and clear indicators of what needed to be done towards project completion. This was highly related to team motivation:

> *Its quite nice to get the focus back on what we are doing and also you know – oh look! we never thought we would get here but we have come this far. It's that feeling of we're doing stuff and also that feeling of ,like, there is only this much to go. And its an physical card so you feel like, you feel like its doable. Or...you can see an end – a deliverable.*
>
> (L.4.60)

A common theme coming from participants as they discussed cohesive, was the importance of being able to see the road to project success. This was seen, not only to increase feelings of security in the project environment, but as a surprisingly strong motivator for action:

> *They thought they knew what they wanted and they wanted a particular date. And I think the...you know it certainly wasn't easy but I think there was a certain amount of...I felt a certain amount of buzz of people feeling that they could get it done, but only just. It was a stretch, but just about possible. I mean earlier on in the project it felt like it was just about impossible. But then, you know, as soon as people started seeing that. You know, as soon as things started to improve and people saw that it was going to be possible, then it was quite a buzz.*
>
> (L.5.20)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The presence of information radiators seemed to allow teams to keep calm and continue to work together in as a team in the face of increased pressure in the project environment. Information radiators displayed things such as the current rate of progress and future rate of progress given the resources available. This makes it clear what actions needed to be taken in order to ensure project success and completion, and provided a focus and solidarity in the team environment that was otherwise missing:

> ***You mentioned that there was one burn-up chart where if you hadn't had that then the team would have fallen apart. I was wondering if you could describe in more detail what it was about the chart that made a difference?***
>
> *Hmmm. Before we applied the burn-up chart there was a lot of stress in the team. And caused a lot of, I think that things got very chaotic. And ah, everything, there was this sense of urgency that was so strong that everyone spent more time working than thinking about how to organize the work. And that lead to a lot of problems. And things taken much longer than necessary, and a lot of not knowing what the state of things were, and a lot of surprises, that things were taking much longer than people thought. And basically what it lead to was that we feel that we lost control of this. We just...we tried to work as hard as we could, but we couldn't say whether we would make it, or what the result would be.*
>
> *(X.3.23)*

Information radiators seen to be particularly important in agile team environments where there were diverging goals, and/or a lack of leaders or focal people to whose job it was to organize and ensure project success. Information radiators were used to give teams a clear overview of the project status which team action could then be organized around:

> *And another thing we were working for three different customers, and um, it wasn't the ah...we lost sight of how much work as to be done for each customer. So there wasn't any person who could say, ah 'Work more for that customer so we are very much in time for this other customer." and I think basically what the burn-up charts did was giving us a very good, very (detailed?) overview of where we are standing. And giving us a sense of control over the situation. Giving us a sense of the things, where we suddenly saw that 'Yes we can make it. There were other things that we already had a feeling that it wouldn't work, but it we couldn't put the finger on it. And the burn chart gave us a way to actually, I don't know how to say, materialize? this feeling. And to get some very simple way to communicate to management. To point to something and say – we have a problem here, and we are not quite sure what to do, and please help us. And I think that's what saved us.*
>
> *(X.3.23)*

Information radiators were also seen as useful in clearly communicating a common goal to all members of a team, which was discussed in previous section to be essential in agile software development efforts:

> ***Saved you from what?***
>
> *From. Ah, I don't know the saying in English. But in German you would say 'drowning in chaos.' I think what was also good was this information radiator, before*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this burn-up, I think we all had this sense of urgency, but everyone was working for himself alone, not as a team. Because we didn't have the time to organize. And the burn chart gave us an opportunity to organize around it. To speak about things, to organize things, to show out in the open. To talk about it, to organize. Yeah, to talk. To make a team again. And to give it a common goal. To make the chart look like it will work. And before everyone has, um; now that I think about it, before hand, everyone had his own tasks, his own stories, and he was concentrating on that and was kind of blindsided. And, um, the information radiator was a way to show that there was a team goal that we had to reach together. Yeah, because the information radiator didn't show individual performance, or individual accomplishment, and it brought focus to the team accomplishment and teamwork.

(X.3.23-28)

Developing a Shared Vision

Working together as a team to develop shared vision is essential on any software project. The number of solutions available to any one problem in software development is immense. One of the common problems faced by participants in this study was the difficulty in aligning ideas of how software should be developed in a team environment. The agile environment was seen to strongly encourage shared understanding through a number of means. Firstly, daily team meetings were seen to greatly increase the chances that everyone was on the page and that miscommunications would be found and resolved quickly. Working software produced on a regular basis also allowed teams to become quickly aware miscommunication of requirements, and provided a tangible artifact about which communication could occur. Communication surrounding a software design was seen to increase the elegance of software solutions and ensure that individually developed parts of the software would fit together and not 'break' other parts of the code. Pair programming, test driven development, and continuous integration were seen as essential practices in this respect:

What happens when they make you stop pairing on a project?

Well you could even see it happening on this project where there was this guy that would just off on his own and do things, and he would break things. And you could see that the stuff he did on his own wasn't as good. And so it took longer in the end.

Why do you think that is?

Well I have been convinced for a while that the main productivity benefit of pairing is that the other person slaps you on the head and tells you 'We don't need that now!' Especially with clever people. Clever coders go of and do clever things, but 'No we don't need it!'

(L.2.17)

Discussion by some participants seemed to suggest that pair programming was seen to allow developers to communicate on a higher level of understanding during coding rather than on a more detailed syntactical level:

The code review itself is via Email. So it's purely textual. That one I think it's a different level. I think a lot of those reviews tend to be more syntax based like you spelt

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this wrong or this looks like it doesn't do exactly like you said it would. It points out, it tries to catch more programming errors. Whereas I think if someone pair programs with you it's more a case 'Of are you going in the right direction?', 'is this...you have kind of made the step but I think you have a longer term view and does that really mesh with the way its going to fully develop in later versions.' So that's more, it's more collaborative.

(T.2.12)

Pair programming was also seen was an essential practice in making sure that individuals on the team would not become overly attached to individual visions of code that did not align with team goals:

If you are reviewing somebody's code that already solves the problem, that you review it and you find that it's adequate but it's a very poor solution to the problem. But it's probably not going to get fixed if it works. Because by the time that it's already done then they have some kind of emotional buy in to the code. They think that they have done a good job now. They have come up with some design. And for one reason or another they haven't come up with the best solution. But they still like the solution that they have come up with. And I mean, everyone is proud of what they do. And so having...if you can bring that review right up to the moment that I first think of a design, before I have had time to fall in love with it, then if someone wants to suggest changes that design. Then preferably there is no code been written, there is nothing to change. But once you write the code and it's already written. Then you don't really take kindly to people saying "well that's shit, you could have done better there.' So um. The earlier in the process you get that feedback, the easier it is to take on board. Just from an emotional point of view, regardless of the practicalities of um, actually having to take a bunch of existing code and change it. It's all about how kindly you take to it.

(L.3.23)

Pairs, it seems, prevent each other from becoming overly focused on solutions that were not productive in the context of the team goals, while at the same time continually affirming team goals in each other's minds:

[When you are coding alone] you are focused, but then your focus starts to head off in the wrong direction. You are still concentrating totally on it, but what you are concentrating on is not necessarily the best track to go down to get the job done. You might have lost sight of the details and be bogged down by the technical details, something like that. Whereas when you are pairing, your pair helps keep you on track. So your focus is actually aimed in the right place.

(L.3.26)

What were the benefits of pair programming for you?

Well let's see. From a personal perspective I find it really helpful that it means that I get someone to sort of go through the thought processes that I am at the same time as I am developing something which is really beneficial when you start going down one path of development and someone says I don't know why you have been doing that. You know you should just do it this way; and it becomes more simpler. And this other person also knows what's going on.

(T.2.8)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The presence of very immediate and clear team goals in terms of the current iteration seems to be essential in allowing team members to maintain a shared focus on the task at hand.

Planning and Estimation

Structured Decision-making

Planning in the agile environment is focused around ensuring that what the team is working on for the current iteration is the most important thing:

> *And if we decide that, you know, feature A is more important than feature B, but feature B is scheduled next, well then feature B gets bumped to the next cycle. And, you know, every month we sort of evaluate what is the most important thing.*
>
> (T.5.6)

The agile planning game provides a very clear structure about which to choose or agree on stories or tasks.

> *And if someone else came up and said I just met with, you know, [a big customer], and they really want this feature. So then the question was, okay 'is that feature more important than something else that's on the board?' If it is then it bumps something else off, you know, versus, sure, we'll put a little bit of it in, we'll put half of it, or a quarter of it in, just to appease [the customer]. You know that's not something that we did, it was like okay is that more important than this blue card, then the blue card gets yanked to the next cycle, you know, and then this [customer] requirement comes in. So it was very much of looking at everything, looking at the resources and identifying how much, you know, how much time we had available and then completing as much of a feature as possible before moving onto a, you know, a similarly significant feature in another cycle.*
>
> (T.5.40)

The planning game seems to give workers very clear parameters about which to base their decisions, and makes it very clear to all parties involved that they are playing a game of trade-off and that the goal of the game is to develop good software. It is especially valuable for team members to be able to walk over to the story board and point to the fact that it is full, and ask the person requesting an additional story to take something off in order for the new story to put on:

> *And the product owner who takes sort of the items that have been called out as priority for the customer and sort of says 'I think this one's next, this one's next, this one's next.' And then the team looks at it collectively and says 'Okay based on our knowledge this is how long it's going to take.' And so there is negotiation back and forth, they say 'Well I want these three' and then the team says 'Well with this sized team, it's going to take us 60 days, we can't do it in that amount of time,' or 'We need to break them out.' So it becomes this negotiation between the representative of the customer and the representative of the product that's going to be built. So it's a very open collaborative effort to decide*
>
> (I.1.14)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Developers seemed to delight in discussing this process because in the past they have been relatively powerless to stop additional features from being added to an already full software project.

Short Term Planning

Long term planning was associated in this study with high investment, increased inclusion of non-essential features, and forced commitment to design decisions that people *knew* would be wrong at a later date:

> *What I have done in the past has been different, like at other companies where it was sort of more waterfall where you created a obnoxious UI stack, you know, from now until next year; and then stuff changed, and I am not a big fan of that. You know there is a lot of investment in terms of writing stuff out and then once you have done all that, the commitment is really high to keep all of that in, even though it may not be the best solution.*
>
> (T.5.5)

The tendency for people, when working in a long term planning situation, is towards inclusion by default rather than exclusion by design. Once requirements are meticulously drawn up in lengthy requirements documents, then there is an investment of time and energy towards these features that makes them very hard to remove, even when it becomes very necessary. Iterations of a week, two weeks, or a month, in comparison, were seen to increase the perception of the current situation as temporary or non-fatal. Data suggests that the short iterations allow for more relaxed team relations because there seems to be less at stake. Cohesive agile teams exhibited less of a need to focus on getting things *absolutely* right, and more of a focus on generating agreement in the group and moving forwards in the process of software development. An interesting finding was that participants were much more willing to put up with less than ideal situations or less than enjoyable work if they knew that it would only temporary.

> *Yeah there are a lot of people in the team that really don't like some of their team members. But let's face it, they can pair with them for a week.*
>
> (O.2.21)

The ability for change and flexibility in the agile environment was seen to greatly reduce the importance of ensuring that personal goals are met, and make people more willing to meet the needs of the team, for the current iteration. This seems to be balanced, again, by the social awareness of individual team member actions, where teams appreciate individual action on behalf of the team.

Another factor associated with long-term development was lack of immediacy. In a software development environment with 6 months left for development, participants related much less pressure to perform, or attend to tedious or difficult tasks. In shorter iterations, the sense of immediacy was seen to increase the willingness of team members to resolve differences of opinion, make decisions, and get on with developing software:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I think also they are more focused when they realize time is running out...I mean when you realize you've only got 4 weeks it will focus you more than 12 weeks, you know there is just no more <waves hands>.

(L.4.47)

Estimation

This study revealed a number of ways in which the task of estimation was made easier in the agile team environment. Firstly, it was made clear that estimates were estimates, and not commitments, reducing the individual stress associated with coming up with a value that they will have to live by no matter what. Various tactics, such as only allowing estimates to be done in multiples (2, 4, 6, 8) were used to drive home this point, fighting a tendency for team members to take estimation more seriously:

And it's interesting because you have to try and resolve that by saying look you know given that we have something similar or given; let's choose a number that indicates how complex this is rather than, you know, getting a written confirmation from you that it will take three ideal days and if it takes any longer you are fired. You know that sort of feeling that from a developer point of view it's quite difficult because it is sort of sense of pride you don't want to say it will take 3 days when it will take 10 days or you know.

(L.4.76)

Estimates were also generally done as a team:

You will go through and confirm the estimates. And what we would do is have a brief discussion with the whole team for 5-10 minutes. And then everyone would do this little game thing where everyone decides what their estimate would be, and then you go like that <shows a hand with two fingers out> and some people would estimate 5 days and some would estimate 2 days, and then we would discuss that. And if I really thought that it would take 5 days then I could stick to my guns about it and if it was my story then it would be put down as 5 days. ...it really comes down mostly; everyone is just reasonable about it. It's not like there is a rule for whose estimate gets in. And everyone knows it IS as estimate. It's not cast in stone that that is how long you are going to take so. But once you have committed to do that story by the end of the iteration, you do work hard to do that story by the end of the iteration.

(L.3.13)

This was seen to generate increased buy-in to the estimates that were initially decided on, as well as a deeper understanding on the part of the whole team regarding what needed to be done in order to complete the task at hand and who was the best person to do it.

An important factor seemed to be that estimates are revisited very close to the time that developers were going to be developing the stories:

Stories are collected and created by business analyst. They will talk to the client and they will come out with high level stories to start with flesh them out in more detail as you come to do them. And there are generally 2-3 levels of detail. You will have a high level thing that you will break further down into stories as it comes to the point where you really need to play that story. Then the developer or the pair who are working on that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

story will go and discuss that with the customer or the BA to find out if there are any further details that they need.

(L.3.17)

As stories are fleshed out in agile software development process, associated estimations are generally revisited. This results in an increasingly accurate estimate as team members consider and discuss the realities of implementation. The estimation process, therefore, ensures that developers have put real thought and effort into developing estimates, as opposed to choosing a number that seems suitable without fully considering the implications or being aware of the exact dependencies involved. Estimation in this manner also supports increased task-based knowledge sharing and well-informed discussion of alternative solutions. This can be held in comparison to a long term planning situation:

Now that we are kind of falling out of Agile a little bit we are starting to, not do waterfall but we are kind of starting to forecast what we are going to be working on in the next year and things like that so. It's putting a bit of a strain on development and us I think because, you know, development needs to identify how long it's going to take them to do something and then we need to get requirements but no one's even thought of it because it's, you know, 12 months from now kind of thing, so we're all kind of figuring out still how we can manage that.

(T.5.17)

Thus agile allows team members to delay discussions and decisions, not only until they have all information at hand, but until they actually *matter* to people. This further allows team members to discuss immediate and important issues that will be occurring next week or next month, and to make commitments that they *know* they can live up to. This was seen to lead to increased feelings of comfort and control regarding what was going on in the software development process, as well as high levels of investment in team goals and activity. Overall, the agile estimation process seems to put significantly less strain on team relations, and to increase developer willingness and ability to complete stories within the estimated time frame.

Prioritization

Putting stories for the iteration in priority order makes very clear the progression of tasks needed in order to get things done. Making this clear and visible to the entire team means that individual preferences and priorities are less likely to control the progress of a project.

[We had] the stories in priority order and I can't remember if we used stickers or ticks or something. Some way of indicating whether they were done or not And also post-it notes with the names of the people working on each task. So you could instantly contact, you know who to talk to about something. And I think one of the things that I was happy with was that, um, that the team hadn't been great at finishing things in the right order. And one of the things that sort of brought a bit of focus was at the stand-up, you know. Asking people how they got on with the tests and so on. And then looking to see what to do next; rather that people saying well I fancy doing this thing here, or you know, thing over here. It would be, well what would be the next highest priority thing; what needs to be done to do that, you know.

(L.5.27)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The fact that there were a relatively small number of stories at the top of the list is also important. There is very clear definition of what needs to be done, and when things are done they are ticked off and put aside. Finished.

Prioritization was highly related in this study to reducing communication overhead – to creating an environment where everyone agreed what needed to be done and did it rather than spending time discussing options or working on different things. One participant described a situation where priorities of someone the team was dependent on were not aligned with the priorities of the team, and the difficulties it caused. He goes on to note:

> *He is not only working for us but for the whole company. So there are things to do for the other team. And there are things to do for management. And he just naturally has different priorities than we have. And he isn't able to...can't understand our priorities and he can't align to him fully. So I think it would be much better if we had a person as a team member who has the same priorities as we have. And who fully works for us as a team.*
>
> ***How would that make a difference in your work, aside from getting things done faster?***
>
> *I think. Um. For one thing it would be much easier to communicate...especially communicate priorities. I assume that if you have someone committed to the team that he would be committed to the team goals, so he simply would understand what priorities we have. And that's currently kind of a struggle.*
>
> (X.3.51)

It was also very important for individuals in that they *knew* that what the team was working on for the current iteration were valuable and important to the customer:

> *So it's, you know, that you are always creating what's going to be the best product and then in terms of what the user wants, because you're evaluating that every month as opposed to a year from now and sticking to it and you are also delivering something that's a lot more complete. So it's a lot more fulfilling in that respect as well.*
>
> (T.5.6)

Iterations and delivery

Team Heartbeat

This is an area that requires future study, but there seems to be important phenomena surrounding the value of the routine of incremental development cycles in the agile environment - which developers discuss as a tangible rhythm or beat – and the ability of teams to organize about this regular cycle of development and delivery. The routine iterations were discussed by developers as related to a sense of team connectedness, and seemed to be both a source of motivation, and a sense of security.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An aspect of the team heartbeat that emerged from this study was the tailored nature of the development tasks for each iteration in teams that allow developers to estimate their own stories. The fact that developers can estimate stories based on their specific skills and motivations seems to support a situation where the team plan for delivery is not asking for more or less than they can comfortably offer. The team pace, therefore, is in synch with developer pace, which may be contributing to the sense of 'flow' felt by developers in the agile environment.

> *There is a pace of development, a rhythm, and you can feel it in the team.*
>
> (O.3.25)

Delivery

The focus on delivery in the agile team environment was shown to be a very strong motivator, particularly amongst developers:

> *I'm in favor of [agile]. For the right kind of people it's a great development method. I don't really want to work in traditional software development teams <laughs>... traditional teams just aren't as focused. In an agile team, because you are working such tight, such short iterations, you are showing something for your efforts fairly regularly, and that's quite a buzz.*
>
> ***Can you describe in more detail this 'buzz'?***
>
> *There is kind of a, a slightly lifted intensity. When you're delivering something every week or few weeks. Whereas in a traditional model you might be going for three months without delivering anything. And when you do deliver something it doesn't often work and there is nothing to show for it because they tend to do horizontal slices. You will develop a component that is useless on it's own, so you have nothing to show for it once you develop it. Agile you are always, you are developing software that people can take it and play with it and do something. Perhaps it is not everything that it should be doing, but it's still doing something that they can see.*
>
> (L.3.1-3)

Data supports the idea that establishing a team history through successful delivery each iteration is essential to the development of trust and respect in the team environment. Rather than having to assume in goodwill that the other team members are doing their part, agile team members are assured time and time again that they and their team members are able to deliver. Delivery of working software was also associated with a sense of importance in the work that was being accomplished and high levels of motivation:

> *We give them a bonus based on the revenue that works out to 2-3 K every 6 months. We though that this would be a strong motivator, but this is not a motivator. It's more like, a thank you that they get.*
>
> ***What is the motivator then?***
>
> *The coffee. The coffee is really great <laughs>. No. I think it is the atmosphere. It's just nice. And the delivery. You are working on something and at the end of the week it goes out to the customer and you get feedback right away. And that's great, because your work matters. Every day matters. You notice when we have a new product and you*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

are working on it for 6 months and then it is really tough going. Because you are not delivering.

(O.2.8)

Delivery every iteration also 'flat lined' the software development process for developers, who were no longer having to work repetitively on similar tasks such as bug fixing for a long period of time:

But with this agile process it's not so much you know, feature-feature-feature development and then bug fixing, you know, and there wasn't this peaks and valleys of really exciting stuff and then bug fixing drudgery. Agile is much more flat lined, so you're always fixing bugs and you're always doing features like, you know, within this...cycle. So, you know, you'll have something interesting to work on but at the same time you're probably fixing a bug, in the previous feature that was just a week or two old. So it's much more – it's much less depressing, you know, than if you have nothing to look forward to than fixing bugs for 4 months.

(T.3.20)

Delivery seems to give team members something to look forward to on a regular basis, which was seen to be highly motivating.

Roles

Understanding Other Roles

One thing that became clear during the course of this study, particularly in talking to a range of roles including developers, interactions designers, and quality assurance specialists, was the huge division between the goals and focus of different roles involved in software development. These divisions were often associated with talk about 'us' and 'them,' and a lack of patience with the other roles and their demands or reluctance to cooperate. Even in teams where understanding between roles existed, there were still points of conflict between roles inherent in the software process itself, which were seen to degrade relationships on the team:

The interesting thing is like it was, I think that there is generally good feeling amongst developers and the project managers and the business analysts and the business, and everyone can see where we're trying to get to, but you do have the traditional sort of - 'What do you want from me?' going on. And that comes from everybody ... a classic one is estimation.

A business analyst will ask you how long a task will take and you have to really work hard to not to...because the question is 'How long do you feel, gut estimate, having seen other things, it will take?' And many people will say 'There is no way I am going to estimate that because I don't have a feeling for it' and etc, etc. And then you can have a sort of almost standoff going on and it's nothing personal but it can be taken personally.

And it's very difficult actually because what you have is like you have people who will feel that things have been taken personally, so they'll say things like 'Chill out' or something, which is obviously not going to work, you know... You do get that odd

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

frustrated look from the BA or the developer going you know; and that is funny because in principle they want to work together, and in practice they will make up and they will try. There is always these other methods that they will try.

(L.4.73...79)

Estimation was seen to be a common point of contention between developers and customers or business analysts (BA's). Agile practices seem to alleviate this tension somewhat by making estimation less binding and only related to the next iteration.

Participants in agile teams with daily stand-ups, shared team rooms, and pairing between roles were also much more likely to show an understanding of the motivations behind other roles in the software development team. This could be compared to participants in non-agile teams who had had less exposure to people in other roles. Such participants would exhibit a general awareness of the pragmatics of other roles, but did not know or show consideration for the underlying motivations of these roles. Interaction designers especially, were seen to appreciate the agile environment for the ability it gave them to share with the rest of the team their purpose in the project environment, and what they were trying to achieve:

I mean each time we are working with people we haven't worked with before and this group was a group we hadn't worked with before we have to go through that little bit of 'yeah you understand what we are doing but you don't really understand it, and so that's you know let's have you watch some of the stuff and we will explain really what we are going to do so that you understand that what we are going to bring to the table.

(T.4.20)

A deeper understanding regarding the motivations and value of the different roles in the software development environment was seen to be highly related to increased collaboration and willingness to work together as a team. Increased interaction and resulting understanding between roles also included a feeling from team members that they were 'on the same boat' rather than working at odds:

It used to be, so for instance before [agile was adopted] ... we would meet with the product specialist once a week and they would tell us what they wanted and what was going wrong and like bugs remaining and so you get the overall impression was like they were demanding, they were always complaining and never satisfied. Whereas in you know, with our culture of communication with them – you meet, and yeah you still get the same kind of information that you know if things are missing or they wanted to go in this direction and not that direction. But you get the underlying feeling that they love the product and that just remains as the bedrock. And then you know on the top you know we can deal with the little bits here and there, the individual features. But I think you can get a better sense of just the communication between the various groups.

(T.2.22)

There also seemed to be inherently different ways of communicating with others associated with roles in the software development environment. The most commonly discussed of these was the tendency of software developers to focus on details and edge cases:

When a software engineer builds a piece of software they have to have a clear idea of what happens in each case. I mean you go down to a machine it has to understand

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in very precise terms. So the software engineer will then ask in very precise terms of the business people, which the business people assume that the software engineer is being pedantic.

(L.1.91)

The agile software development was seen to increase exposure to individuals in other software development roles, and therefore increase understanding of other roles, and how best to interact with them:

You know that's, you get a sense of just how to interact with people. And before that I think it was much more separate...the developers knew each other pretty well and how they talked to each other. But the product specialists who sort of defined the direction of the features we talked to them maybe once a week. And so you don't get a sense so much on how they communicate or what they complain about, or, you know. You get a different impression.

(T.2.21)

The agile planning game especially was seen to increase the amount of understanding regarding the constraints of each party, and make inherent conflicts of interest easier to digest:

And then the other thing where there is a lot of tension is just when you develop a feature there is only so much you can do, you can't...well the product specialist will often come in with a list of a 1000 items on it. And the developer has...there is this dance that takes place and the developer estimates, they say well I can do 50 of those. And you know that basically boils down to the acceptance criteria that gets used and you know frequently the acceptance criteria will include things that just impossible to do. And so we will have discussions about that a lot earlier and that also causes a little bit of tension because you know, now the product specialists realize the implications of all they are asking for. And then likewise the developers realize that they actually do need to put some thought into, how they are going to do something so that they can provide reasonable feedback and reasonable estimates and things like that.

(T.1.11)

The visibility of these effects through the use of information radiators was seen as especially important in bringing home awareness of constraints and in reducing attempts to squeeze more features into a software development product than could be plausibly developed:

So those themes can change but we try to keep the goal there because it often will map to what the user is expecting in that release, and so from there we have our goals or our themes for that release and now we have all these features that are scattered across the board and we want to make sure that the tasks or the features in that release meet that goal or are keeping with that theme and that with those estimates on each of those individual tasks we can tell, well we've only got five weeks but we've got ten weeks worth of work to do here so something's got to give. We take five of that and we push it to the next cycle or, and re-adjust that theme or, you know, everything has an effect and you can see that effect on the big board when you do the planning.

(T.3.18)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Effect of ease of change on team dynamics

Working in an environment that makes necessary changes easy was seen to greatly effect software development team dynamics. Relationships between roles were not as strained, since change requests are not seen as bothersome demands but as requests for product improvement. Team relations in agile teams were seen to be generally more friendly and amiable, with a high level of receptivity to requests or suggestions coming from the other roles or individuals:

> *So for myself I always found that it really helpful because what we would do is we would have like a big cycle planning where we would spend half a day with development and the product specialists and sort of all of the people; the stakeholders. And it would be identified what we were going to be doing in the next cycle ... And then that would give me an opportunity and the person that I work with, because we work in pairs here, so it would give me an opportunity to, you know, start gathering the requirements and the design and, you know, be getting the design and then testing it and iterating it and by the time that design was done, the development team was ready to start working on it.*
>
> *So it actually fostered a really nice relationship between development and [the interaction designers], because when you're testing and you need to make changes and you are kind of running back and forth to development, depending on how much investment development has put into the code, it can be really bothersome for them to have to keep changing things. So, you know, from this perspective it was really nice because, you know, they could make minor tweaks to the prototype or we may have even been testing with paper at this point. So the, you know, the investment by them was a lot less so they would essentially, we have to do things once and it was done. So there was a really good dynamic there, they were always very receptive to what we had to say because, you know, we weren't bothering them all the time. And they were really good about saying 'Okay, you know, what do you think about this?' or stuff like that, so it was just a very friendly dynamic.*
>
> (T.5.3)

Holistic Involvement and the Creation of Quality Software

It is unclear whether the increased involvement from developers in this study was due to the nature of developers recruited and present in teams doing agile software development, or whether agile practices themselves increased involvement. Still, the kind of sharing between roles and involvement of team members across the entire project seemed to result in increased ownership, understanding, and involvement in the whole team and the whole product, as opposed to their individual parts. This, in turn, resulted in increased initiative for further team communication:

> *And I will always present designs as well to the development team and sort of get feedback that way and then they are very involved as well when we are gathering requirements. So before we start a design, you know, we'll invite the development -- working on the feature to join in and the product specialist will join and they will essentially, you know, talk to us about what their requirements are and so development is there to hear it right ... so there is a lot of communication, a lot of involvement.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(T.5.11)

This could be held in comparison to environments where team members were responsible for only part of the software process, which were associated in this study to loss of control, annoyance at not being able to follow through, and increased friction between roles:

And do you think working in a more waterfall-based environment affected your dynamic with developers?

Well, it certainly wasn't as close-knit because essentially you created a huge document which was very time consuming, and you essentially handed it off to someone else who, you know, took it over; who started implementing it in terms of being able to use the usability test and verify your UI design. Once you have handed over that spec it was very hard to get changes made. So there was the lot of resistance from development and, annoyance with our group; and for us it was unpleasant too because you want to, I mean, I always felt that I wasn't doing my job as well as I could have been because I was delivering stuff that I knew hadn't been verified and it just, it wasn't a good feeling because I knew that ... I can't predict a year from now what some is going to want to use and how it is supposed to look because it really depends on the other features you are putting in because you want everything to work nicely together and look the same and stuff; So it was very different.

(T.5.6)

Members of an agile project are much more likely to be assigned full time, and so be involved in the entire development process. Such involvement was seen to allow increased focus and immersion in the project at hand.

Does it make a difference being part-time or full-time on a project?

Makes it easier. The people on my team much prefer to work full-time, to be a full-time member and to be able to get really deep on one application. That's what they have told me. And at one point we actually tried taking one of them and splitting him to two, on two because we felt; it was one of those, let's us just try this okay to see if it works or not. And it was a failure. It was terrible because he spent so much time context switching between the two projects he couldn't do the amount of work he needed to do on each. And so he felt like that he wasn't working 50% on both he was working 20% on both and the rest of the time was taken context switching between the two projects.

(T.4.33)

Another factor related to being full time on the team was better team relations resulting from increased familiarity, and the fact that team members can rely on each other without having to compete against time spent on other projects.

Having all roles present in the agile environment also allowed team members to collaborate throughout the software development process to achieve their goals. This was seen, not only to make their jobs much easier, but to increase the chances that team members would identify and fill gaps in the teams software development efforts and produce a higher quality product:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

With the traditional development we would never have like QA and doc people talking to us all the time. They would be put on right at the end. And I really thought that was fantastic. So again as an interaction designer we would have the documentation person write the docs while we were creating a prototype that create prototype docs and we test those together whereas we normally could never do that. And the QA person was sitting there the whole time like listening to what we were talking about, the problems we were finding and so they could do QA type stuff. Or we were doing interaction design type testing and the two would mesh nicely ... it was nice to be able to talk to them make sure they were covering everything.

(T.4.27)

As far as the developers went, I don't know, I found the developers much more interested in the project and much more interested in what we were doing because the tight feedback loop, they could actually see the difference. Like they could, I think most developers want their stuff to be successful. It's just that they are normally not given the information to make it successful. Like they don't want to make a bad UI, right - they want to make a good UI, they are just not given the information. So when we give them the information they could change something and then we could come back and say, 'The users are – they want the prototype, they don't even want to wait. They kept trying to pull it from us right.' That makes them happy too. So we had a nice relationship there that we could give them the feedback that they were really, that they really wanted.

(T.4.28)

The previous segment also illustrates the value of positive feedback in the agile team environment. This can be held in comparison to the kind of feedback which exists in a many software development environments, which one participant described as:

Not much at all unless there is something wrong with it, and even then it will probably be to late to change anything.

(O.3.31)

Another theme that came out of discussion with participants in agile software development teams was the value of being able to 'do things properly,' instead of just hacking things together. There were a number of reasons given for this: team members had more time to do a good job; engaging in incremental improvement over iterations; the safety net provided by agile practices such as automated testing ensured quality code; and the ability to easily get information needed from other members of the team:

How does being in an agile environment affect you as an interaction designer?

...it affects me by making me be able to do what I am supposed to do. Like what I am supposed to do is be the voice of the user and try to get the interface to do what our goals were and if the Agile process allows me to do that by allowing me to get the feedback in and actually act on the feedback then that makes me happy. I am a much more satisfied interaction designer because I am actually hitting goals that I want.

(T.4.26)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Whatever the reason, the presence of a quality product, and knowing that they were doing the best job possible was seen to be very important to team members in software development teams:

> *I guess the other thing is just it was nice knowing that we were always ahead of development, so the pressure was off a little bit to, you know, to get stuff done right away, you could really take your time and make sure that things were done correctly.*
>
> (T.5.4)

The value of agile in allowing team members to produce quality software was related to the product as a whole as well as to individual tasks:

> *You know, so that we had, a lot of things were very, if not entirely complete before... we kind of lived in that every cycle the product could potentially ship as a complete product, it may not have had a complete workflow, but the features that were in it, were complete.*
>
> ***Why was that so important?***
>
> *Well, it was that in the past, sort of pre-Agile, we had some products that went out that were, that should not have gone out, you know, and that kind of stuff it can really effect your reputation as a company. You know probably most of the features were incomplete, so there wasn't a complete workflow, and the way that product had been developed, you know, they would do bits and parts of these features, but because it was waterfall implementation QA wouldn't get it till the very end and then, you know, they're potentially testing, they're testing things at the very end potentially missing stuff, at that point there is not enough time anyway to come back and fix bugs...*
>
> *(T.5.41)*

Process Improvement

Room for Improvement

Many agile teams discussed during this study engaged in process improvement, where the team would work together to improve how they functioned to develop software. Highly functional and cohesive agile teams discussed with participants in this study were usually engaged in process tweaking to some degree, usually by way of retrospectives to ensure team buy-in to changes. Process improvement was highly related to individual motivation and enjoyment in the agile team environment, as well as the ability for teams to function as cohesive units.

One of the more valuable aspects of agile software development methodologies highlighted by this study was that they allow and encourage such process improvement. This can be compared to non-agile organizations where individuals wanting to instantiate change are often stymied by barriers to process improvement coming from their organization, management, or peers:

> *For me it's always a discussion in my organization is that – what do I take for granted, and what do I try to change. Because I'm not the kind of guy that is so visionary*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

or zealot like that I want to fight windmills all the time. And ah, so I do pragmatically what I can do. I am a pragmatist. So I do what I can do, and try to be a bit pushy once in a while.

(X.4.33)

Agile as opposed to non agile teams also exhibited an increased awareness of how the team worked as a whole, and an increased awareness of improvements that could be made. For example:

We once had a time where builds would take more than two hours. And in that time, there were times when we actually didn't notice that the build wasn't running any longer. It took us days to notice that there hadn't been a build for days. And so that was a strong indicator for us, actually, that not caring about the feedback was wrong. We forgot about the feedback. About how important it is – the cycle wasn't working anymore because it was too big.

(X.3.14)

This seemed partially related to the fact that there is an agile ideal that team members are aware of and trying very hard to live up to:

Yeah. So using the XP, well Agile way. So that's sort of whatever things you know. Although we tried it and that was good that we tried it. We didn't quite abandon pairing, we didn't abandon testing, didn't abandon any of those things. But we didn't seem to feel like we got it fully previously and I think that's probably because we had different systems and different code and this has allowed us to focus more on how we would like things to be

(L.4.7)

Non-agile team members, on the other hand, generally exhibited a lack of comprehensive awareness of process and an increased focus of personal rather than tasks and issues.

Part of what was allowing process improvement in the agile environment seemed to be the safety net of regularly delivered software and ability to easily revert back to previous ways of working if the current iteration is not a success. Agile environments were also associated in this study with increased team involvement, interaction, and empowerment, which seems to increase motivation, and the ability of individuals to improve their working environment.

Tinkering

Iterative development in agile development supported cyclical process improvement, and a kind of 'process tinkering' was seen in agile teams as each iteration allowed progressive optimization of team processes:

To some extent it is up to, ah, developers are good at problem solving, and also business analysts are also good at problem solving by, almost facilitation. So helping you find the way to solve your problem. So you've got two skills there that if you work together you can probably say well you know last week's situation kickoff didn't go too well, we almost felt mauled. You know one side of the other might feel mauled at the end of it and so. You know you might come out with a better iteration kickoff than, I mean we

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

> *have taken some time to get to a balance where the iteration kickoff is not like 5 hours long where everyone is pooped and wants to just go home and you know hate iteration kickoff's. To... like having -- which is like okay it's very easy to estimate but no-one is questioning this and there is that sort of you know.*
>
> (L.5.80)

Tinkering was especially popular with developers, who seemed to very much enjoy the excitement that came from playing around with how things worked in the team environment. Tinkering in the agile project environment allowed teams to suit the agile process to their individual needs for communication and collaboration, and seemed to increase the chances that agile teams would achieve the state that participants described as 'really working together.'

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.