MANAGING EXTERNAL INNOVATION: THE CASE OF PLATFORM EXTENSIONS

by

Nadea Saad Noori

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Technology Innovation Management

Department of Systems and Computer Engineering

Carleton University

Ottawa, Canada, K1S 5B6

November, 2009

Your file Votre référence
ISBN: 978-0-494-63813-2
Our file Notre référence
ISBN: 978-0-494-63813-2

ABSTRACT

In recent years, high-technology industry witnessed the emergence of a product system where a focal company provides core components (a platform) and external parties provide complements. Increasing user demands for new and customized features product require companies to seek external sources of innovation. To align external innovation with company goals and protect the platform integrity, companies need to take a more active role in coordinating participation of external parties and controlling the quality of innovation outcome.

This research examines strategies adopted by platform owners to manage the quality of platform complements developed by external parties. The research is focused around software platforms and one type of complement: platform extensions.

This research helps to understand strategies followed by platform owners to manage developing platform extensions in two areas: i) governance, and ii) regulatory instruments.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Prof. Michael Weiss, for his tireless guidance and support throughout the process of researching and writing this thesis. Without his support, I would not have been able to pursue my dream of a master's degree. I will always be grateful for that.

I would like also to thank Professor Susan Logie at the School of Linguistics and Language Studies (SLaLS) for her support and guidance in the process of writing and reviewing this thesis.

To my husband Marc and my daughter Marjane, who gave me love and strength to complete my graduate program.

To my mother, Musharaf and my three sisters Shaimmaa, Noor and Hiba in Iraq and the USA, who always supported me and encouraged me to pursue my dreams.

Last but not least, to my class mates and friends in the TIM program, for the endless hours we spent preparing assignments and presentations for our courses and which prepared me for the work of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

OSGi Architecture	The OSGi architecture is a set of specifications that define a dynamic component system for Java. These specifications enable a development model where applications are (dynamically) composed of many different (reusable) components. The OSGi specifications enable components to hide their implementations from other components while communicating through services, which are objects that are specifically shared between components.
POJO	The Plain Old Java Object is the name used to emphasize that the object in question is an ordinary Java Object, not a special object and in particular not an Enterprise Java Bean (EJB). It was coined by Martin Fowler, Rebecca Parsons, and Josh Mackenzie (Jamae & Johnson, 2009).
UNO	The Universal Network Object is component-based model for the OpenOffice.org. It offers interoperability between programming languages, object models and hardware architectures, in process or over process boundaries, as well as in the intranet or the Internet. Uno components may be implemented in and accessed from any programming language for which an Uno implementation (AKA language binding) and an appropriate bridge or adapter exists.

1 INTRODUCTION

Iyer (2006) describes how software companies are operating in a small world of interconnected networks and innovation is increasingly taking place in such networks. The question is no longer whether or not to open the innovation process and collaborate, but how to best leverage a network of external parties (Tuomi, 2002; Moore, 2006; Pisano & Verganti 2008; Vujovic & Ulhoi, 2008). Collaboration allows companies to share risk and cost, gives them access to new markets, resources, and knowledge, and reduces the time to develop a new product (von Stamm, 2008).

At the core of each innovation network is a focal organization known as platform owner (or keystone that provides the platform to facilitate contribution by other members in the network (Iansiti & Levien, 2004). In our research, we define a platform with Hagiu et al. (2009) as a product, service or technology that provides a foundation for other parties to develop complementary products. This platform can be owned or by a single player, as in the case of Apple's control over the iPod and iPhone application ecosystems (Shuen, 2008), or be developed and influenced by a group of players, as in the case of the Eclipse software development platform (des Rivieres & Weigand, 2004).

The literature on how platform owners manage complementary markets is focused on complements that built on the platform but not on complements that integrate with the platform such as extensions. The quality of platform extensions affects the overall

performance of the platform and other complements (Messerschmitt & Szyperski, 2003).

Platform extensions are the complements that extend the functionality of a platform beyond its core capabilities. Extensions can integrate in different levels with the platform and other existing extensions and this integration reflects upon the platform integrity. Managing the quality of platform extensions had become a concern for platform owners because: i) the value of a platform is realized as the value of the system comprising the platform and complements (Gawer & Cusumano, 2008), and ii) the quality of extensions affects platform integrity.

In this research we explore an issue that has not received sufficient attention in the research literature, although it is of great practical importance. We examine the strategies followed by platform owners to adopt open innovation concepts in the sense of opening up the platform to external parties and how to manage the quality of complements developed by those parties. We limit the scope to software platforms and to one type of complement that is platform extensions because of the direct effect of the quality of extensions on the platform integrity and to keep the study to a manageable size. The initial results of our work were presented at the 3rd FLOSS International Workshop on Free/Libre Open Source Software (Noori & Weiss, 2009).

1.1 Objectives

The objective of this research is to answer three research questions:

1. What are the models used by software platform owners to coordinate external innovation and manage the participation among the community of developers of platform extensions?

2. How do platform owners encourage and benefit from external innovation in the platform network while protecting platform integrity?

3. What is the impact of the platform extension management approaches on innovation levels in the platform network?

Opening up the platform to exploit higher levels of innovation in the platform ecosystem entails a risk for the platform quality and its complements (Thomke & von Hippel, 2002; Gawer & Cusumano, 2008; Boudreau & Hagiu, 2009; Hagiu, 2009). Boudreau and Hagiu (2009) present evidence on the role of platform owners as regulators to protect the platform from negative effects such as low quality complements. To reduce the negative impact on the platform quality, platform owners have a number of regulatory instruments to regulate the platform ecosystem such as pricing, toolkits, and design rules (Baldwin & Clark, 2006; Hagiu, 2008; Bosch & Bosch-Sijtsema, 2009).

This research is focused on the process of developing platform extensions, the strategies followed by platform owners to encourage external innovation, manage participation, and to control the quality of extensions developed by external parties as well as the tradeoff between innovation levels and the quality of extensions.

1.2 Deliverables

The deliverables of the research are:

- Framework that platform owners have adopted to manage the process of developing extensions and encourage the innovation in their network or ecosystem.

- Propositions that describe relationships between the governance models adopted to manage the process of developing extensions, the extensions quality, and the flow of ideas in the platform network.

- Propositions that describe relationships between regulatory instruments, extensions quality, and the flow of ideas in the platform network.

1.3 Relevance

There are at least three groups who may be interested in the outcomes of this research. First, potential and existing platform owners who need to understand what strategies other platform owners have adopted to control the quality of extensions and why.

Second, third party developers can have different roles in the process of building platform extensions, as they develop the extensions and build toolkits to harness platform development. Therefore, third party developers need to understand the tools and means provided by platform owners to build and control the quality of platform extensions.

Third, this research will provide students and researchers with new research opportunities to develop models for managing external innovation in platforms.

1.4 Contributions

This research contributes to our understanding of the models followed by platform owners to encourage external innovation and manage participation in their network in the case of platform extensions in two areas: i) governance models, and ii) regulatory instruments.

The research explores the lever points that enable platform owners to control the process of developing complements (i.e. extensions) and manage participation among the network of developers.

1.5 Organization

This thesis is organized into seven chapters. Chapter 1 presents the introduction. Chapter 2 covers the review of the relevant literature. Chapter 3 presents the method used to conduct the research. Chapter 4 describes the cases of software platform in the

sample. Chapter 5 represents the research results and these results describe the governance models and regulatory instruments used to manage extensions quality. Chapter 6 represents the discussion of the results, where relationships between governance, openness, extension quality, and flow of ideas are organized in two sets of propositions. Finally chapter 7 provides the conclusions, describes the limitations of the research, and identifies opportunities for future research.

2 LITERATURE REVIEW

We divide the review of the literature into two streams: open innovation and collaboration strategies (Section 2.1), and software ecosystem and platform strategies (Section 2.2). Section 2.3 presents the lessons learned from the literature review. Table 1 represents a summary for the literature review streams and some of the reviewed literature for each stream.

Table 1. Summary of the literature review streams

No	Stream	Rationale	Literature
1	Open innovation and collaboration strategies	• Examine different approaches adopted by organization to open up the innovation process • Examine different approaches to control the collaboration in the organization network • Extract the dimensions for the governance models and the community architecture of participation	Chesbrough (2003) Vujovic & Ulhoi (2008) West & O'Mahony (2006) Pisano & Verganti (2008)
2	Systemic innovation and platform strategies	• Understand the concepts behind the platform -complement product system • Examine the different relationships and information transactions among the members of the platform ecosystem • Examine the importance of the platform owners role to regulate the platform ecosystem • Examine the different platform regulatory instruments and how each affects the information transactions in the platform ecosystem	Baldwin & Clark (2006) Bosch (2009) Bosch & Bosch-Sijtsema (2009) Boudreau & Hagiu (2009). Gawer & Henderson (2007) Maula et al. (2006) Thomke & von Hippel (2002) von Hippel (2001) West & Gallagher (2006)

2.1 Open innovation strategies and collaboration governance

This stream has two sub-streams. First we examine existing literature on open innovations strategies to gain an understanding for the different approaches to open the innovation process and how different strategies affect the flow of ideas and the level of innovation in the organization network. Second we examine the collaboration governance and architecture of participation literature to explore the different governance models adapted by the organizations to control the collaboration in their network and identify the dimensions of the governance models.

2.1.1 Open innovation strategies

Chesbrough (2003) portrays a fundamental shift in innovation strategies for companies that he calls open innovation. Companies no longer rely exclusively on internal innovation, but also incorporate ideas that originate outside the company. He notes that "the boundary between a firm and its surrounding environment is more porous, enabling innovation to move easily between the two" (ibid, p. 37). This shift also implies that companies no longer control all aspects of the innovation process. In exchange, companies can learn about opportunities that fall outside their current dominant logic.

Despite the claimed benefits for opening the innovation process, it is not an easy task for any organization and the complexity of the process depends on the strategy adopted (Vujovic and Ulhoi, 2008). Vujovic and Ulhoi (2008) defined four types of

strategies for opening the innovation process based on the type of user involvement and the organizational level at which co-operation with the external resource takes place.

The first strategy is a closed circle approach where a number of organizations voluntarily exchange information in a joint effort of problem-solving. Channels to share the knowledge are formal and protected (i.e. professional meetings, associations, and journals). The second strategy is the open-source innovation approach where the community contributes to a pool of innovation and the knowledge is shared back with everybody. In this strategy, the organization would mediate the innovation and information sharing among the community. Also to ensure that the knowledge will not be restricted in any way, there exist different legal means like licenses for that purpose. The third strategy is to combine closed and open innovation modes. In this case a company combines an exclusive proprietary business model with a non-proprietary one. IBM is an example as it has its own open-source development services and has the commercial propriety software solutions as well. The last strategy mimics the open-source innovation strategy as the organization opens some communication channels with users, but the knowledge contributed stays within the boundaries of the organization (Vujovic and Ulhoi, 2008).

2.1.2 Architecture of participation

Architecture of participation plays an important role in managing the contribution and leveraging innovation in the platform network as it provides a guide for the interaction and exchange activities in a community through social, legal, and technical means offered to its members. The architecture of participation presents a socio-technical framework that assists the platform owners to extend the participation opportunities of external parties and integrates their contribution (West & O'Mahony, 2008). The platform architecture, production process and intellectual property rights are subsets of the community architecture of participation that enable platform owners to leverage innovation and control the collaboration process in their network (West & O'Mahony, 2008).

2.1.3 Collaboration governance

Establishing a governance structure is an important asset for the platform owners as it will leverage the interaction in its network or ecosystem (Hagel III et al., 2008). Pisano & Verganti (2008) explore how innovation leaders leverage the collaboration in a network of external parties. Their framework identifies two dimensions of collaborative innovation: how open or closed membership in the network of collaborators, and how flat or hierarchical the governance structure of the network should be.

Openness encourages the flow of lots of information between the platform owner and external parties, but the challenge is filtering the large flow of information without the burden of high screening cost. On the other hand, having a closed network could provide the platform owner with the best sources of knowledge but the challenge is to identify the right source.

The governance structure determines who sets the direction of innovation and who appropriates the value created from the innovation. A hierarchical structure enables the owner to control the direction of the innovation but the identifying the right direction could be challenging. Therefore, having the capability to understand the user needs and provide systems that enable the contribution of external parties help the platform owner to choose the right direction. With a flat governance structure, everybody in the platform network contributes to the innovation process, which lifts the burden from the platform owner, but it might be challenging to get the community to converge on a useful solution to the platform owner. However, having processes and rules in place could help external parties to achieve common goals.

Pisano & Verganti (2008) framework entails distinct trade-offs between governance structure and network openness. This distinction results in four basic models of collaboration as shown in Figure 1. The elite circle model (closed, hierarchical), where the platform owner is in total control of its network as it selects the network members, problems and solutions. The consortium model (closed, flat) operates like a private

club where problem selection and solution selection are a joint effort among the consortium members. In the innovation mall model (open, hierarchical) the platform owner identifies the problem, the community proposes the solutions, and the owner chooses the best solutions. The last model is the innovation community model (open, flat) where the process of problem definition, solution offerings, and choices are made by the community (Pisano & Verganti, 2008).

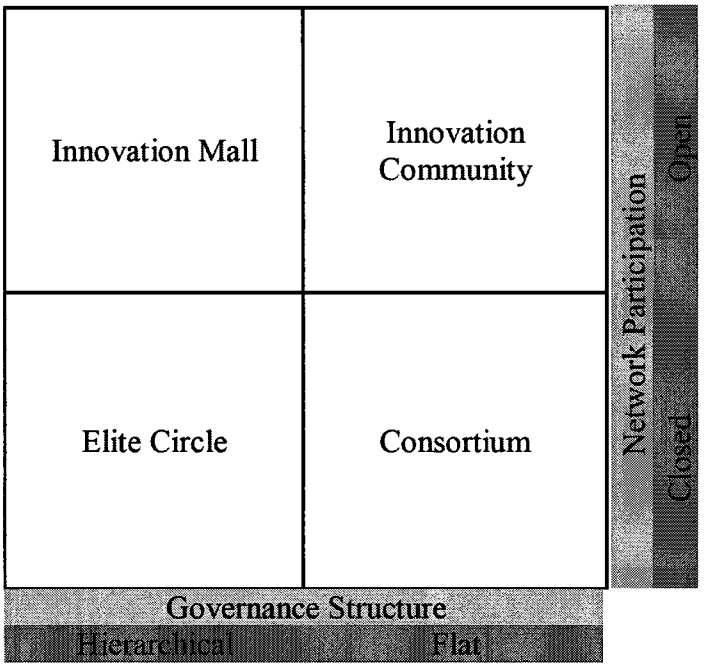

Figure 1. Collaboration governance model by Pisano & Verganti (2008)

2.2 Systemic innovation and platform strategies

In this section, first the systemic innovation literature is reviewed to extract the importance of the quality of the platform-complements product system as a whole. Second, software ecosystems and platform strategies literature is reviewed to gain an

understanding for: the relationships within the software ecosystem, information transactions among the ecosystem members, and the structure of the platform-complement product system.

2.2.1 Systemic innovation

Innovation has become increasingly systemic as identified by Maula et al. (2006). In a systemic innovation world, focal companies provide the platform for innovation and they depend on external parties to provide necessary components that complement this platform. Integrating systemic innovation concept within the organization strategy reduces the cost of coordination and provides control benefits, but it has been seen that many firms lack adequate resources (e.g. financial or technical) to create the necessary environment for successful systemic innovation (Maula, 2006).

Systemic innovation had become a collaborative process where the focal company (i.e. platform owner) and the external parties create a resource pool for parties working on different components of the systemic innovation that are the innovation platform and its complements. The focal companies had become more dependent on the innovation in complements to support their platform; therefore, the external parties are crucial for the platform success (West & Gallagher, 2006). In order to reach these external resources, the focal company must demonstrate a credible commitment to the systemic innovation process and there must be significant benefits to both parties (Cusumano &

Gawer, 2002). Although the focal company provides the architecture of the systemic innovation, the external parties are not under the direct control of the focal company.

Systemic innovation involves a shift from an internal innovation process to an open and collaborative process. This shift in the innovation process created the need to find new governance modes and tools to carry out activities related to the creation of a successful systemic innovation.

2.2.2 Software ecosystems and platforms

A platform can be defined as a product, service or technology that provides a base on which other parties to build complementary products or services that run on top of the platform and extends its functionality (Boudreau & Hagiu, 2009). The Software ecosystem consists of a software platform that is surrounded by a community of consumers and complementors (i.e. supplier, competitor, or partner). In the ecosystem, a platform value increases with the value of its complements (i.e. extensions, application, service, hardware parts, etc) (Bosch &Bosch-Sijtsema 2009; Chesbrough, 2003).

Figure 2 illustrates a simple software platform ecosystem. At the center of Figure 2 resides the platform owner, which is a firm that owns or sponsors the platform core and plays the keystone role in platform ecosystem as well as influencing the platform forward evolution (e.g. Google, Microsoft, IBM) (Gawer & Henderson, 2007; Iyer & davenport, 2008; and Iyer et al. 2006).

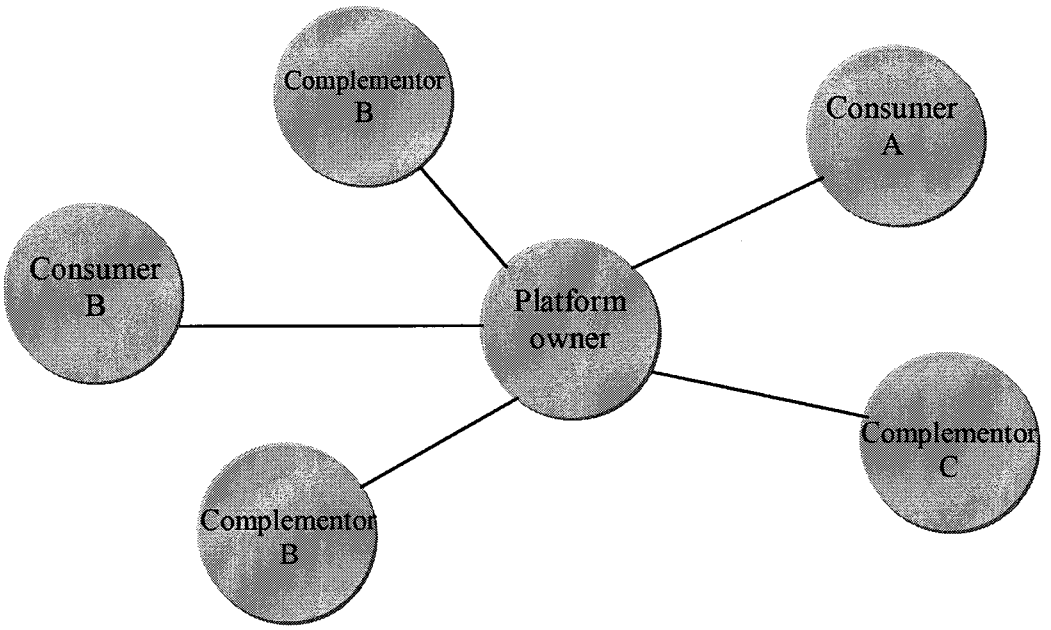

Figure 2. A simple software platform ecosystem

The platform owner has an essential role to facilitate the interaction among the different ecosystems members as well as providing the platform infrastructure. Platform owners have at least three motivations to regulate connections among members and control the flow of ideas in the ecosystem. First, the platform owner wants to minimize issues associated with R&D cost, product complexity, risk mitigation, coordination problems. Second, the platform value lies in the ecosystem wrapped around it, therefore, the platform value increases if it has a high quality complements. Third, platform owners want to leverage external innovation to achieve a specific goal. These motivations will reflect on the platform owner approach to

regulate the relationships with external parties (Boudreau and Hagiu, 2009; West 2008).

There are different approaches when it comes to opening up the platform for development by external parties. A directed or close circle approach implies that the platform owner opens up specific areas to be developed by selected external parties. The undirected or community approach implies that the entire platform is made available for external parties to contribute and the platform owner has nothing to do with selecting what part of the platform to develop or what functionality to build. The last approach is to have multiple tiers of internal and external developers (Bosch, 2009).

2.2.3 Platform architecture

The platform architecture describes the structure of the platform components, their interrelationships, and the principles and guidelines used to extend the platform. The platform architecture has an important role in leveraging external innovation, reduce the investment and efforts for R&D, and accelerate the innovation in the platform ecosystem (Hagel III et al., 2008). Also the platform architecture affects the community participation as suggested by West and O'Mahony (2008).

Therefore, the platform architecture is an important element in the strategy of opening up the innovation process in two ways. First, the platform architecture would leverage the contribution of third-party innovators and enable them to rapidly add new products

and functions. Second, the platform architecture enables the platform owner to control the flow of information and helps monitoring the innovation activities in network. The Google's innovation ecosystem is an example of an ecosystem that is built around a proprietary platform of computing infrastructure to support Google's search operations. The infrastructure comprises services such as Google Ajax Search, AdSense, and Maps. Google exposes these services by defining a set of interfaces that are known as Open API. Third parties can use these Open APIs to create mashups that incorporate these services. This architecture enables Google to leverage third-party innovation while maintaining architectural control. Google also gains access to valuable analytics and to ideas for new applications and improvements to its services than it could not have discovered internally (Iyer & Davenport, 2008).

Modular platform architecture enables and increases the involvement of external parties in the platform development process as the modular architecture concept provides:

- Supporting a good development methodology by having different parts of a large design to be worked on concurrently. The independent blocks in a modular structure can all be worked on simultaneously.

- Managing complexity by limiting the scope of interaction between components, which means reducing the amount and range of cycling that occurs in a design or production process. As the number of steps in an

interconnected process increases, the process becomes increasingly difficult to complete successfully and the time spent increases; therefore, the probability of success and the output quality may decrease. Having a modular design reduces the range and scope of potential cycles and this limitation results fewer cycles and less time spent on the development process. As a result modularity will increase the probability of success and the quality of the final output.

- Accommodating uncertainty by partitioning the design parameters into those that are visible and those that are hidden. Hidden parameters are isolated from other parts of the design, and are allowed to vary. Modular designs are flexible therefore; if new knowledge later yields a better solution for one of the hidden modules, then it will be relatively simple to incorporate the new solution without having to change the rest of the system (Baldwin & Clark, 2000).

So the concept of modular architecture spans an important set of principles: design rules, independent task blocks, clean interfaces, nested hierarchies, and the separation of hidden and visible information. The modular platform architecture helps to focus development efforts on developing specific modules without having to learn the whole system. As well, it enables platform owners to keep control of the platform core architecture that is a powerful barrier against competing architecture and protects the integrity of the platform itself (Cusumano & Gawer, 2002; Messerschmitt & Szyperski, 2003).

In a modular architecture, the modules developed have to be composed because a single closed software component offers less value than one that can be combined with other components to achieve greater functionality. Composability requires two properties: interoperability and complementarity.

Interoperability means that modules must be able to communicate by having communicating infrastructure, a protocol used to communicate, and mutual communicating encoding. The communication mechanism enables the modules to complement each other in terms of what functions and capabilities they provide and how they provide them.

Performance is an important aspect of software composition because we can have two separately fast components, but they can be very slow when combined. So integrating different components affects the overall system performance (Messerschmitt & Szyperski, 2003).

The platform consists of a core that provides common functionalities and extensions that implement additional functionality. A platform extension is any module that is added after the platform deployment (Messerschmitt & Szyperski, 2003; Bosch, 2009). An extension could be a minor change such as adding a button to the toolbar, or a major change such as adding a new feature that extends the core functionalities (Baldwin & Clark, 2006; Gawer & Henderson, 2007). It is worth mentioning that the application programming interface (API) is an open interface designed to accept a

broad class of extensions, where extension means a module or modules added later. Extensions are not only modules added following the modules development but following modules deployment as they extend from the full knowledge of existing modules and add new capabilities to the platform. The quality of the developed modules is measured by performance and usability, both of which are two aspects of module composability (Messerschmitt and Szyperski, 2003). Extensible platform architecture empowers platforms because of the ongoing input from the community to complete and enhance the platform (Hemrajani, 2006b).

Platform extensions may have different forms such as plug-ins, add-ons, and modules. Extensions can be classified as: external extensions, internal extensions, and extensions that become part the platform core. Figure 3 represents the platform architecture with different types of the platform extensions.

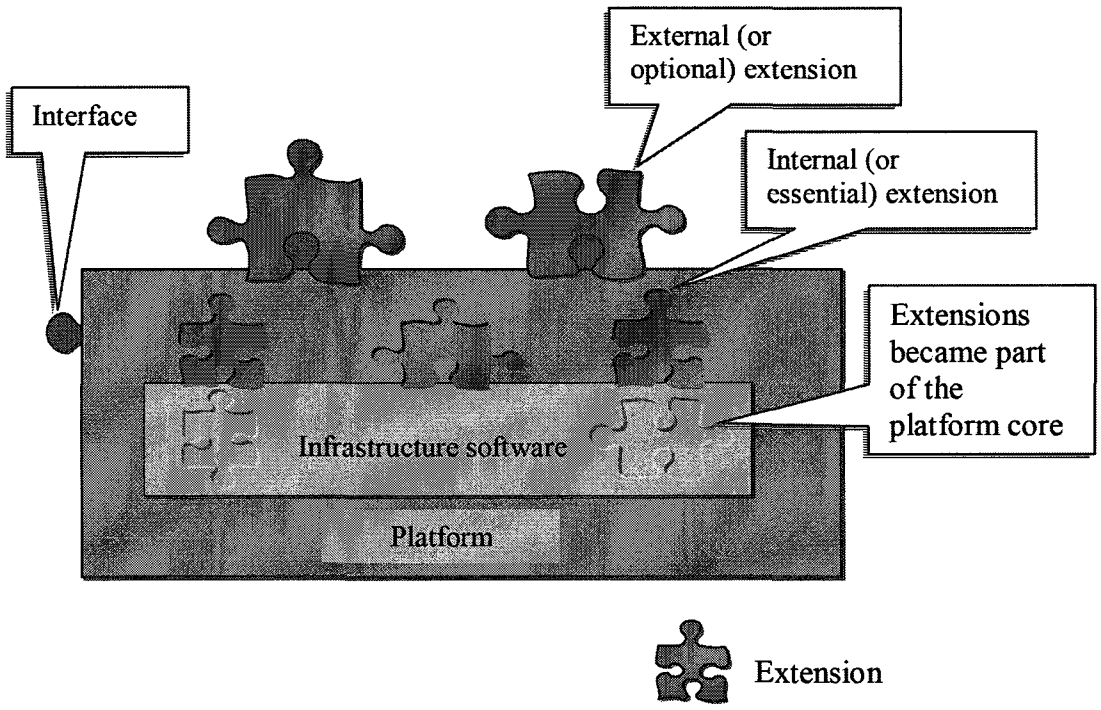

Figure 3. Platform architecture and types of extensions

The distinction between external and internal extensions originates with the microkernel architecture implemented by systems that need to be made available in multiple configurations that share common core functionality (Buschmann et al., 1996). Microkernel architecture has a minimal runtime (corresponding to the platform core) that provides hooks for adding extensions. All required functionalities are provided in the form of internal extensions that are integrated with the platform, while application-specific functionality may be added through external extensions. Internal extensions can remain extensions or can be incorporated into the platform core (Messerschmitt & Szyperski, 2003; Baldwin & Clark, 2006; Gawer & Henderson, 2007; Bosch, 2009).

Extension status can change from external to internal or from internal extension to fully integrated with the core, as a platform evolves. This transition is not random, but usually follows a process controlled by the platform owners for managing extension development. One example is the Tabbed Browsing feature of the Firefox web browser, which was initially available as an external extension, but was later bundled as part of the Firefox core platform (Noll, 2007).

2.2.4 Regulatory instruments

A platform ecosystem consists of the platform provided by the keystone, and a community of complementors. This community includes developers that build applications that sit on top of the platform, modules within its core, or components that extend platform functionality (Bosch & Bosch-Sijtsema, 2009). Boudreau and Hagiu's (2009) study presented evidence of the important role that platform owners play as a regulatory body within their network or ecosystem. Platform owners play the role of public authority, and they impose regulations to reduce the negative impact on the quality of the platform. There are number of instruments that are called "regulatory instruments" to help platform owners enforce regulations in their ecosystem (Boudreau & Hagiu, 2009).

Regulatory instruments are a menu of options that platform owners use regulate relations among members in their network, such as pricing, platform design and legal means (e.g. licensing, developers' agreement, etc) (Bosch & Bosch-Sijtsema, 2009;

Boudreau & Hagiu, 2009). The regulatory menu of instruments includes: pricing, legal and contractual instruments, platform architecture, toolkits and sandboxes (Boudreau & Hagiu, 2009; von Hippel; Bosch & Bosch-Sijtsema, 2009).

In our research we focused on instruments that platform owners are offering for extension development and quality assurance, such as toolkits (Thomke & von Hippel, 2002; Messerschmitt and Szyperski, 2003; Boudreau & Hagiu, 2009). In the next five sections we are going to discuss the different types of instruments in general, such instruments include: pricing, legal and contractual instruments, platform architecture, user toolkits, and sandboxes.

Pricing and membership

Pricing and membership are two forms used to regulate information access and transactions on the platform. Pricing or membership refers to the pricing schema set by the platform owner to charge third-party developers for gaining access rights to the platform, information, and service (Hagiu, 2008). Although there might be a large number of developers willing to pay to gain access to the platform resource, platform owners are likely to exclude low-quality developers to protect the quality of the platform (Hagiu, 2009).

Legal and contractual instruments

Legal and contractual instruments are largely used to protect the contents of the platform and its components and reserve the rights of both platform owner and developers. Legal and contractual instruments are licensing and property rights and other traditional legal rules that are established between the platform owner and the third party developers to manage access to the platform, information, and services. An example is MySQL, which uses the Sun Contributor Agreement[1].

Platform architecture and design rules

The platform architecture is a set of rules that define how platform modules communicate with each other and how to communicate with elements in the platform environment (e.g. applications, external components, etc). The platform owner defines the core design of the platform and the set interfaces that force boundaries among users and developers. The platform architecture enable the platform owners to define "an appropriate solution space" as von Hippel describes it, where the manufacturer (i.e. the platform owner) offers the user a certain degree of freedom to develop products within the production system (von Hippel, 2001). The platform architecture provides platform owners with a mean of virtually controlling the third party activities

[1] http://forge.mysql.com/wiki/Sun_Contributor_Agreement

by imposing laws and design rules (Messerschmitt & Szyperski, 2003; Boudreau & Hagiu, 2009).

User toolkits

In the traditional product development model, the interface to the customer is the product prototype, and feedback on how well customer needs are met is only obtained late in the product development cycle. In the user innovation model of product development, the locus of innovation shifts from the company to the customer (Thomke & von Hippel, 2002). The new interface to the customer is now a platform that customers can adapt to create solutions to their needs (Thomke & von Hippel, 2002). The iterative experimentation needed to develop new products is now carried out by the customer. This results in significantly faster cycles and reduces cost. The creation of solutions is supported by a toolkit that allows users to carry out some of the product development tasks themselves. Toolkits present another form of software infrastructure that reduce the time and effort required to develop, provide, and operate applications, and contribute to quality of the product. Also the toolkits' design plays an important role in increasing the opportunity to generate ideas in the community (Messerschmitt and Szyperski, 2003; Jeppesen, 2005).

Mediating toolkits also play a significant role in ensuring quality, as identified by Rossi & Zamarian (2006). These toolkits crystallize effective solutions to recurring problems, and allow these "filtered" experiences to be shared. One motivation to use

toolkits is the complexity of the platforms, and to hide low-level aspects of the platform from extension developers. Prügl & Schreier (2006) distinguish between low-end and high-end toolkits; depending on the scope describing the design possibilities users can explore using the toolkit. They found that in addition to the toolkits offered by the platform owner, which tend to be more basic, users have created more advanced toolkits to meet the needs of specific user groups.

Providing toolkits is not restricted to platform owners, as users develop toolkits that serve the users' needs in the process of developing elements for the platform. An example is the K Desktop Environment (KDE) that was developed by Matthias Ettrich (Spinellis & Gousios, 2009a).

Software sandboxes

The software sandboxes are another form of user toolkits that can be provided for innovation. Traditionally sandboxes defined as a controlled environment that provides at least a minimal functionality needed to accurately test the developed software (Venugopalan, 2005). Sandboxes provide an isolated environment for untested code changes and an experimentation space away from the original code-base repository. There is no uniform definition of what a sandbox comprises. A testing server, a mirrored production environment (working directory), and a development server are different formations of sandboxes that exist today.

In the context of web application development, a sandbox is a staging or development server that is logically separated from the production server. There is even a more specialized notion of sandbox in the sense of an environment for executing non-trusted applications (such as applets).

A sandbox can also include the capability to test multiple platform configurations where the module being tested is combined with other modules and the opportunity to simulate environmental conditions to exert the module (for example, if deployment of the module requires the presence of special hardware, or should be tested with multiple users) (Bosch & Bosch-Sijtsema, 2009).

2.3 Lessons learned

The lessons learned from the literature are:

- The research regarding the development of platform extensions is still in its early stages, so there is not much literature on the strategies and frameworks used to manage the platform extension development process.

- Open innovation leads to porous boundaries between firms and their surrounding environment, which facilitates information flow with external parties, but also results in a loss of direct control over the quality of product features.

- Elements of the architecture of participation such as technical design, development process and intellectual property are important enablers for leveraging innovation and controlling quality in the platform network.

- There are different models to open up the innovation process and collaboration governance but there are no frameworks to control the quality of the outcome.

- Contribution of external parties is crucial to the platform success because platforms are becoming more dependent on complements and the platform value lies in the value of its complements.

- Products, services and technologies can become a platform for other parties to build complementary components, applications, services, which either run on top of the platform or integrate within the platform, to extend the platform functionality.

- Platform owners play the role of public authority and they impose regulations to reduce the negative impact on the platform quality.

- Platform architecture leverages the contribution of third-party innovators by enabling contributors to rapidly add new products and functions as well as it enables the platform owner to control the flow of information and helps monitoring the innovation activities in network.

- Flexibility of the modular platform architecture empowers platforms because it enables the integration of the ongoing input from the community to completes and enhance the platform.

- Platform extensions are modules deployed to extend or add functionalities to the platform and they integrate in different level within the platform.

- Regulatory instruments are important to enforce regulations among the members of the platform ecosystem to enhance product quality and leverage innovation in the platform network.

- Firms are recognizing the power of user innovation and of collaboration with external parties, and are trying to leverage resources and competences outside the firm by providing well designed innovation toolkits.

- Innovation is shifting from the company to the customer; toolkits help to increase the opportunity to generate ideas in the platform network.

- Toolkits mediate quality by making experience filtered down from repeated use of the platform shareable.

- Sandboxes help developers by providing the capability to test multiple configurations and the opportunity to simulate environmental conditions that might be hard to replicate.

3 RESEARCH METHOD

This chapter describes the method used to produce the deliverables of this research. It is organized in three sections. Section 3.1 describes the unit of analysis. Section 3.2 identifies the study period. Section 3.3 describes the research method and the steps undertaken to answer the research

3.1 Unit of Analysis

The unit of analysis is a software platform that provides the ability for external parties to extend its functionalities. Platform extensions can take several forms, for example plug-ins, add-ons, modules and new features are considered extensions. The research focus is on extension that adds a new feature to the platform and as a result will affect the integrity and over all performance of the platform. Firefox tabbed browser or Apache Application Server new core module are examples of extensions affect the core functionality of the platform.

3.2 Study period

The data were collected from March to July 2009. The cases covered in this research are between the years 2000 and 2009. The reason behind the decision of choosing this time period is that several platforms that provided extensibility were launched (e.g. Eclipse platform, OpenOffice.org, and SpringSource.org), some platforms were re-factored or re-written to support extensibility (e.g. Apache, IBM WebSphere, and

MySQL) (See Figure 4, the case platforms timeline). Platforms up to June 2009 were covered in this research to establish a limit for the researcher's data collection.

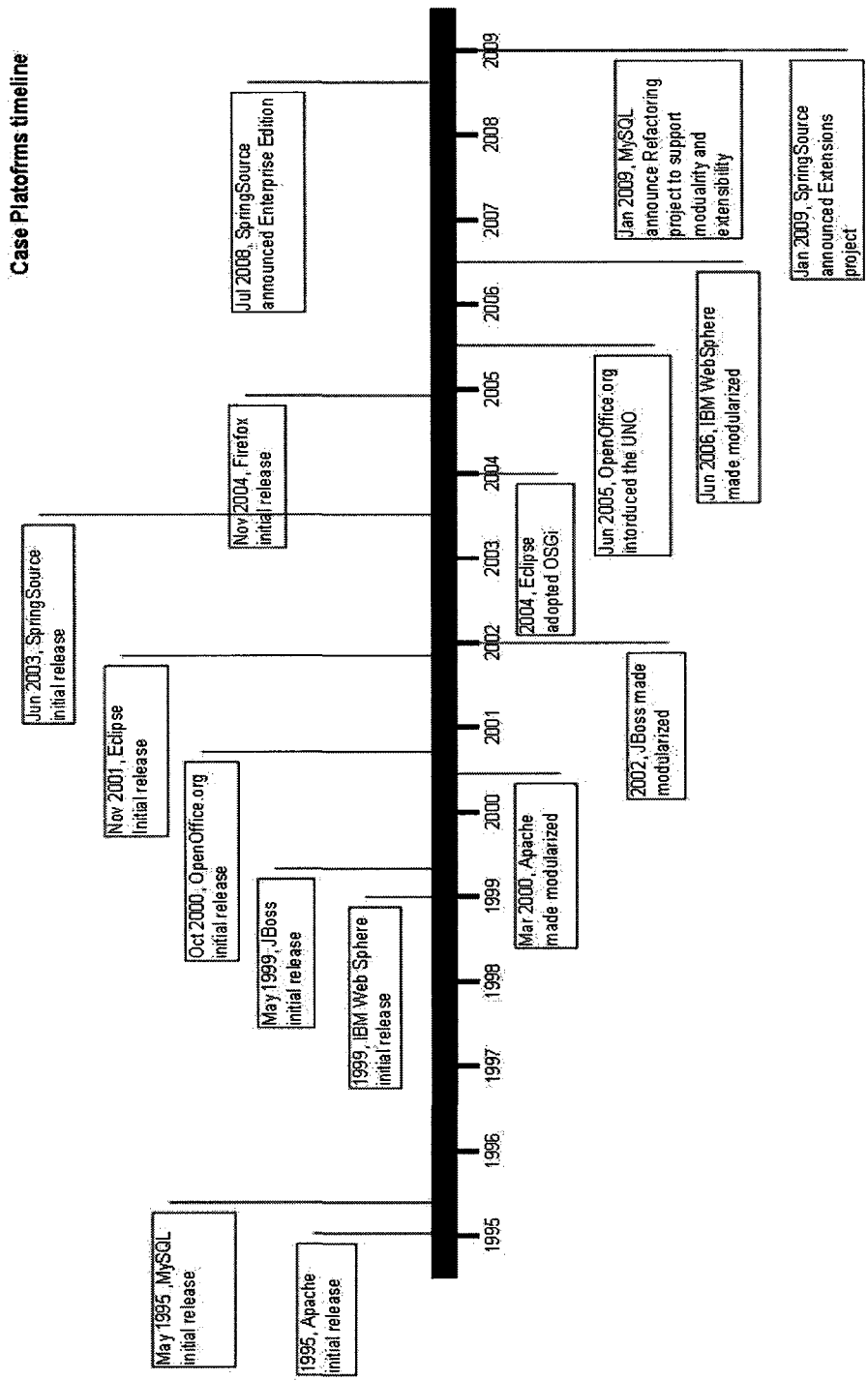

Figure 4. Case platform timeline

3.3 Method

The research around regulating the platform extension development process is still in its early stages, therefore there is little known about the strategies adopted by platform owners to control the quality of extensions developed by external parties (Hagiu, 2009). We used the case study research approach due to the novelty of the research area (Eisenhardt, 1989).

Table 2 provides the steps of the research method. For each dominant activity in the method, the rationale for the activity is identified in the last column of Table 2.

Table 2.Research method

No	Activity	Rational for activity
1	Define objectives for the research	• Focus efforts and provide references for construct measures • Set goals for research completion
2	Extract the dimensions for the collaboration models dimensions and platform regulatory instruments from the literature	• Specifications for collaboration models dimensions and the instruments used to regulate platform networks
3	Identify sample of extensible platforms	• Define population of software platform to draw samples
4	Select the first wave of platforms randomly and collect data	• Random selection was used because previous research did not provide a reference to select the initial cases using

No	Activity	Rational for activity
		theoretical sampling
5	Analyze data	• Categorize the different approaches followed by platform owners to control the process of developing extensions • Examine the instruments utilized by platform owners to regulate the network of extension developers
6	Select second wave of platforms randomly and collect data	• Random selection was used because previous research did not provide a reference to select the initial cases using theoretical sampling
7	Select third wave using theoretical sampling and collect data	• Focus on cases that show potential to extend or refine initial findings
8	Identify strategies adopted by platform owners to control process of extensions development	• Answer the research three questions

3.3.1 Define objectives for the research

The objective of the research is to examine strategies followed by platform owners to manage the external innovation in the case of platform extensions. In order to achieve that goal we need to answer the following questions:

1. What are the models used by software platform owners to coordinate external innovation and manage the participation among the community of developers of platform extensions?

2. How do platform owners encourage and benefit from external innovation in the platform network while protecting platform integrity?

3. What is the impact of the platform extension management approaches on innovation levels in the platform network?

3.3.2 Extract dimensions to describe the platform extensions quality control strategy from the literature

Table 3 presents the criteria used to establish differences between the different strategies followed by platform owners to control the quality of extensions developed by external parties. Table 4 presents the criteria used to choose the factors to measure levels of innovation and quality for the platform.

Table 3. Criteria for establishing differences in strategies used to control the quality of platform extensions

Dimension	Qualitative value of the variables	Criteria
Platform architecture	• Core • Internal extensions • External extensions	Platform components can be segregated to core and extensions or core, internal extensions, and external extensions.
Governance structure	• Hierarchical • Hierarchical/flat • flat	Platform owner is controlling the community contribution or everybody can contribute
Network openness	• Open • Closes	Platform owner selects developers network members or everybody can join the network
Community structure (Developer's role)	• Core • Internal (Strategic) • External (Independent)	The segregation levels of the developer community for the platform extensions
Regulatory instruments	• Pricing • Non-pricing (i.e. toolkit, sandbox, etc)	Instruments that are utilized by the platform owner to control the quality of contribution

Table 4. Criteria used to measure factors affected by the strategies used to control quality of platform extensions

No.	Dimension	Qualitative value of the variables	Criteria
1	Flow of ideas	• High • Medium • Low	Number of contributors or partners and number of modules developed or published by the platform community or development activity
2	Extension quality	• High • Medium • Law	Number of extension download, number of the extensions downloads, or users reviews, or extensions ranking, or bug reports

The following dimensions where selected from the open innovation and the collaboration governance literature: i) governance structure, ii) network openness, These dimensions were extracted from the open innovation, architecture of participation, governance, and collaboration literature (Pisano & Verganti, 2008; Vujovic and Ulhoi, 2008; and West & O'Mahony, 2008; Hagiu, 2009) and were selected because they are used to distinguish between different community participation architectures and different collaboration governance models in networks and ecosystems.

The following dimensions were selected from the systemic innovation and platform strategies literature as well as the open innovation and the collaboration governance literature: i) regulatory tools, ii) platform architecture, and iii) role of developers. They were extracted from the architecture of participation literature and the platform strategies literature (Thomke & von Hippel, 2002; Messerschmitt & Szyperski, 2003; West & O'Mahony, 2008; Bosch, 2009; Boudreau & Hagiu, 2009). These dimensions were selected because they present different means that help platform owners to control the quality of extensions developed by external parties.

The dimensions for measuring the effect of strategies used to manage participation are selected from the open innovation and the platform strategies literature: i) flow of ideas, ii) extension quality, These dimensions were extracted from the open innovation and the platform architecture (Chesbrough, 2003; Messerschmitt & Szyperski, 2003;

Pisano & Verganti, 2008) and they were selected as flow information and ideas presents the innovation in a certain network and extension quality is the other factor affected by strategies adopted to manage the participation.

3.3.3 Identify sample of extensible platforms

A list of software platforms was created by using information available on the internet. After the list was created the following steps were undertaken to select the sample:

Step1:

Refining the resulted list to include only extensible platform by using keywords like "extensible, extensions, extension" in combination with the platform name.

After the list was created another round was conducted to filter out any result that does not fit within the extension definition (see section 2.2.1). To decide which platform to discard, the researcher examined the different sections on the platform website (like developers guide, community, how to contribute, press release and other) and other available resources from the literature.

Step 2:

The platform website was examined to determine if the platform was made extensible between the period of January 2000 and June 2009. Platform website sections "home", "about", "history", "frequently asked questions", "press releases", and mail

archives were used to draw this information. A list of eligible extensible software platforms was produced to be part of the sample.

3.3.4 Select waves of software platform cases

Cases were selected in three waves and Table 5 shows the 12 cases that were included in the sample and selected cases for each wave. For the first and second waves, four cases were chosen randomly from the refined list of software platforms that was produced in step 3. Eisenhardt (1989) suggests using theoretical sampling when selecting cases for case study research which implies that cases may be chosen to replicate previous cases or extend emergent theory. For this study, previous research did not provide a reference to select the cases using theoretical sampling as suggested by Eisenhardt (1989). When data from the first two waves was collected and preliminary results emerged, a third wave of four projects was chosen using theoretical sampling. The last four projects selected had the potential to extend and refine the initial results (Eisenhardt, 1989).

Table 5. Platform cases included in the research sample

No	Platform	Wave	Institution hosts/ owns the platform	Platform website
1	Eclipse platform	1	The Eclipse foundation	http://www.eclipse.org/platform/
2	IBM WebSphere Application Sever		IBM	http://www-01.ibm.com/software/webservers/appserv/was/
3	Mozilla Firefox browser		Mozilla Corporation and Mozilla Foundation	http://www.mozilla.com/en-US/firefox/firefox.html
4	Spring framework, open source		VMWare, Inc.	http://www.springsource.org
5	Spring framework, commercial	2	VMWare, Inc.	http://www.springsource.com/products /enterprise
6	Apache HTTP Server		Apache Software Foundation (ASF)	http://httpd.apache.org/
7	IBM HTTP Server		IBM	http://www-01.ibm.com/software/webservers/httpservers/
8	OpenOffice.org		Sun Microsystems, Inc.	http://www.openoffice.org/index.html
9	StarOffice	3	Sun Microsystems, Inc.	http://www.sun.com/software/staroffice/
10	JBoss Application Server		Red Hat, Inc.	http://jboss.org/jbossas/
11	JBoss Application Server, enterprise edition		Red Hat, Inc.	http://www.jboss.com/products/platforms/application/?s_kwcid=TC|8574|application%20server||S|p|3689179731
12	MySQL Database Server		Sun Microsystems, Inc.	http://www.mysql.com/

3.3.5 Collect Data

Building platform profile

For each platform we build a profile based on the information available on the platform website and from the literature. The platform profile consists of 4 sections: platform architecture, collaboration mode, structure of the extensions developer community, and the platform regulatory instrument. Table 6 shows platform profile sections and the source of data for each section.

Table 6. Platform profile sections, description and data source

No.	Section	Description	Data source
1	Platform architecture	Represents a brief description for the platform extensible architecture and the different integration levels of extensions	Platform website pages like: platform architecture, overview; also the literature
2	Collaboration mode	Represents the approach the platform owners controlling the collaboration in their network. It has two dimension: governance structure and network openness	Platform website pages like: development guide, proposal process, project creation guidelines; also the literature

No.	Section	Description	Data source
3	Community structure	Represents the extensions developer community structure depending on the set of rules set by the platform owner to segregate the community	Platform website pages like: governance documentation, governing entity charter; and the literature
4	Regulatory instruments	Present any regulatory instrument use by the platform owner to control the quality of third-parties developed components	Platform website pages like: toolkits, products, QA page; and the literature

3.3.6 Analyze data

Platform extensibility

In order to determine the levels of the platform extensibility and modularity; a brief description was created for the platform architecture and the different integration levels of extensions. Data extracted from the platform website and the literature about the platform architecture was organized to: i) describe the features of the platform architecture, ii) describe the principles for the platform extensibility, and iii) identify the platform extensions types.

Collaboration model

Information was drawn from the website of the platform cases in the sample. The website sections of the platform case such as "developers centre", "how to contribute", "community", "members", "get involved", "projects", and "frequently asked questions" were examined searching for information about the collaboration model (Pisano & Verganti, 2008) adopted for each type of extensions.

The collaboration model suggested by Pisano & Verganti (2008) has two dimensions: governance structure and network openness (as described in section 2.1.3). The governance structure has three levels: i) hierarchical, where the platform owner imposes restrict rules and exercise high influence on the extensions development process, ii) hierarchical/flat, where the platform owners enforce less regulation and have a medium influence on the extensions development process, and iii) flat, where no restrictions are imposed and the platform owner has no influence on the extensions development process. The network openness presents the level of contribution of the extensions developer community compared to the platform staff. There are two levels: i) open, where contribution is made by all community members, and ii) closed, where the contribution is limited to the platform staff (i.e. core developers).

Community structure

Information was drawn from the platform website sections about governance documentation such as governing entity charter, or contribution guidelines, or project proposals, or membership levels, or how to contribute pages. Data extracted represents the extensions developer community structure depending on the set of rules set by the platform owner to segregate the community. There are three types of developers as described in the Table 7 and Figure 5 shows the position of the different developer types in relation to the onion model to present the community structure.

Table 7. Types of developers for the platform extensions developer community

Type	Code access rights	Scope of work
Core	View and submit code to the platform codebase and other modules*	Platform core components
Internal (Strategic)	View and submit code to the internal extensions modules only*	Internal extensions
External (independent)	View and submit code to the external extensions modules only*	External extensions

* For open source projects everybody has the right to view the entire project source code.

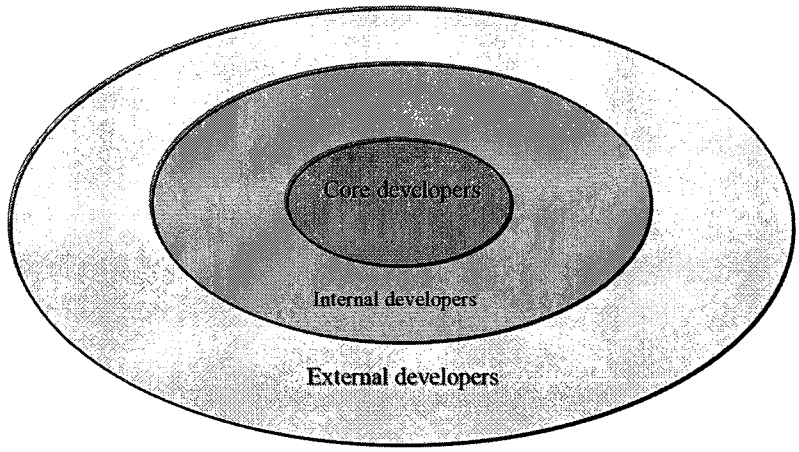

Figure 5. Developer community structure and developers types

Regulatory instrument

Regulatory instruments data was extracted from the platform website sections related to the different types of regulatory instruments. Website pages like "how to contribute", "membership", "partners", "sponsorship", and "community members" were examined to search for pricing structure or membership fees. Web pages like "development toolkits", "sandboxes", and "quality assurance" were to search for tools provided by platform owners to help test, integrate, and publish different types of extensions. The technical design instrument type data extraction was covered in the platform architecture section. The data about the extensions development was extracted from website pages like "project proposals, "project incubator", "extensions project", and "frequently asked questions". As for the legal and contractual instruments, sections about user and developer agreements were investigated.

3.3.7 Identify strategies adopted by platform owners to control process of extensions development

Governance models used to govern the extensions developer community

Identify the governance model for each case based on the following factors: platform architecture, collaborating model, community structure. The governance model represents a socio-technical framework for the extensions developer community where the platform architecture covers the technical aspect f it; and collaboration model and community structure cover the social aspect of the framework.

Regulatory instruments used to control the quality of extensions developed by external parties

Organizing the different instruments that was extracted from the platform website pages and examine the instruments that were used to harness the extensions development process particularly and control the quality.

Governance models and regulatory instruments affect on innovation levels and extensions quality

We refine the Pisano & Verganti (2008) framework to adapt it to quality management. Our framework has four independent dimensions: governance structure, network openness, platform architecture, and regulatory instruments and two dependent dimensions: extensions quality and flow of information.

Based on the above, two set of proposition were constructed. The first set describes the relationship between the governance models and the extension quality as well as the flow of ideas. The second set describes the relationship between the regulatory instruments provided by the platform owners and the extension quality as well as flow of ideas.

4 WRITE UPS FOR PLATFORMS IN THE SAMPLE

The sample used in this study included 12 platforms that have an extensible architecture and were published between the years 2000 and 2009.

For each case we focus on the extensible architecture of the platform, the extensions development process, the extensions community of developers, and tools employed to support the process of developing extensions. Each case profile covers the platform architecture, collaboration mode, architecture of participation, and regulatory instruments.

Table 8 presents a list of the platforms included in the sample. For each platform Table 8 provides: i) wave in which it was included in the analysis, ii) the institution/organization that hosts or owns the platform, and iii) platform's website.

Table 8. Platform cases included in the research sample

No	Platform	Wave	Institution hosts/owns the platform	Platform website
1	Eclipse platform	1	The Eclipse foundation	http://www.eclipse.org/platform/
2	IBM WebSphere Application Sever		IBM	http://www-01.ibm.com/software/webservers/appserv/was/
3	Mozilla Firefox browser		Mozilla Foundation	http://www.mozilla.com/en-US/firefox/firefox.html
4	Spring framework, open source		SpringSource.org	http://www.springsource.org/
5	Spring framework, commercial	2	SpringSource.com	http://www.springsource.com/products/enterprise
6	Apache HTTP Server		Apache Software Foundation (ASF)	http://httpd.apache.org/
7	IBM HTTP Server		IBM	http://www-01.ibm.com/software/webservers/httpservers/
8	OpenOffice.org		Sun Microsystems, Inc.	http://www.openoffice.org/index.html
9	StarOffice	3	Sun Microsystems, Inc.	http://www.sun.com/software/staroffice/
10	JBoss Application Server		Red Hat, Inc.	http://jboss.org/jbossas/
11	JBoss Application Server, enterprise edition		Red Hat, Inc.	http://www.jboss.com/products/platforms/application/?s_kwcid=TC|8574|application%20server||S|p|3689179731
12	MySQL relational database management system		Sun Microsystems, Inc.	http://www.mysql.com/

4.1 Eclipse Platform

The Eclipse platform project and the core projects are managed by the Eclipse foundation. The fist release for the Eclipse platform was in 2001 and it has evolved from an integration platform for tools (des Rivieres & Wiegand, 2004) to a general platform for integrating applications and services (Gruber et al., 2005).

Platform architecture

The Eclipse platform is based on a plug-in architecture. The platform consists of a kernel known as "Platform Runtime" that provides tools for extension management. All the platform functionalities, including the "core" functionality such as basic UI elements, are implemented in the form of extensions and are referred to as "plug-ins" in the Eclipse environment. A plug-in is the basic distribution unit of functionality in Eclipse. Each plug-in plugs and declares its dependencies on other plug-ins. The plug-in declares extension points, and implements extensions to the extension points of other plug-ins (des Rivieres & Wiegand, 2004). Figure 6 presents an overview for the Eclipse platform architecture.

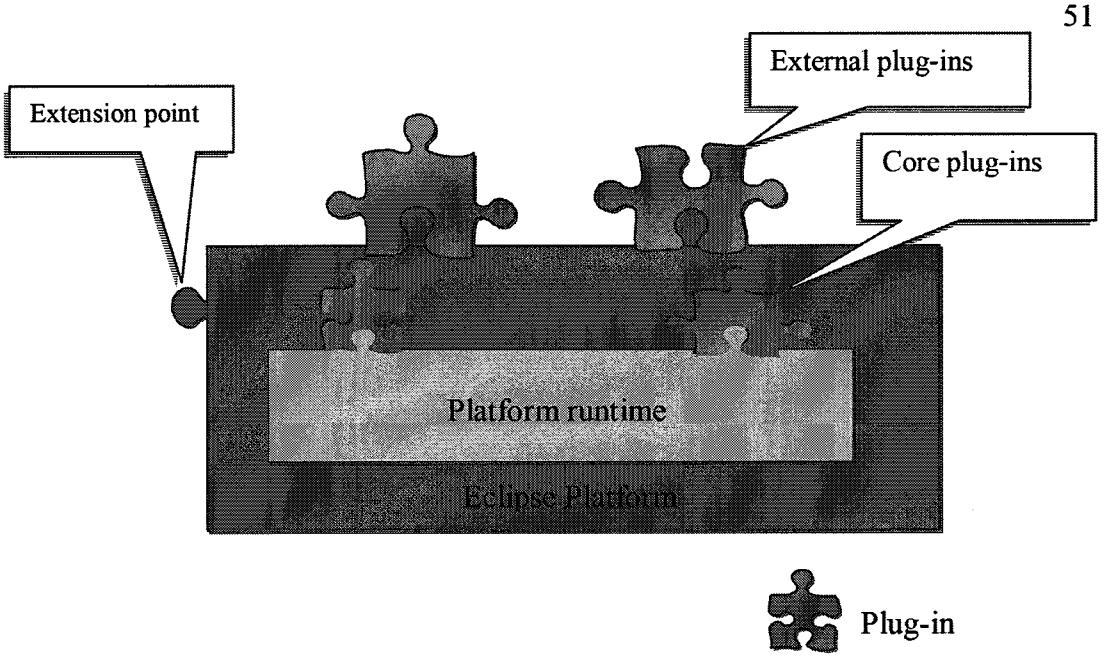

Figure 6. Eclipse platform architecture overview

In 2004 the Eclipse platform core was re-implemented as an OSGi- compatible service container, the Eclipse 3.0 or Eclipse Equinox. The Open Service Gateway Initiative was founded in 1999 and its mission is to standardize the services for local networks and embedded services. Services based on the OSGi open standard specifications can be executed on OSGi complying servers. Eclipse 3.0 consists of the core runtime below it sits the OSGi Service Platform that provides the core models for Eclipse components and services. In Eclipse 3.0 all components above the OSGi Service Platform are bundles including the Eclipse Core Runtime platform which allowed multiple runtime configurations by bundling different components. Eclipse

implementation helped to separate the development of software components from their customization and assembly. Also the OSGi architecture supports the dynamic configuration of systems meaning plug-ins can be added and removed without having to restart the Eclipse (Daum, 2004; Gruber et al. 2005).

Collaboration Mode

The Eclipse platform is a subproject among others in the Eclipse project. The Eclipse project is organized as a set of core projects, which are featured prominently on the Eclipse website[2] (Duenas et al, 2007). Each project is implemented as a set of plug-ins and they are the equivalent of internal extensions. Regular plug-ins are listed on repositories such as Eclipse Plugin Central and they are equivalent of external extensions. Since the focus of the Eclipse platform is to serve as platform to facilitate the development of commercial products, commercial plug-ins (for example, the IBM Rational Application Developer for WebSphere application server, or the Oracle JBuilder IDE) have been available as of early on. Hence, in the Eclipse platform, only the quality of core projects and commercial plug-ins is tightly managed. Table 9 represents the different collaboration modes of extensions for the Eclipse platform.

---

[2] Projects Summary-Eclipse Project.
(http://www.eclipse.org/projects/project_summary.php?projectid=eclipse). Retrieved Oct 9, 2009.

Table 9. Collaboration modes for Eclipse extensions

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open		Eclipse core projects	Eclipse plug-in
	Closed	Eclipse platform core components and Commercial plug-ins		

Community structure

The Eclipse project high-level structure defines two types of project: top-level projects that are associated with core components development; and standard projects that are associated with internal extensions development. The top-level projects have several projects and one Project Management Committee (PMC) with executive authority. Standard projects have a project lead and community of committers and developers. Decisions are made by voting, the PMC handles decision making for the standard projects and the Eclipse Management Organizations (EMO) handles decision making for top-level projects. As for regular plug-ins that presents the external extensions for the platform there is no formal process to control the quality, but there are other means as discussed later in the regulatory instruments section.

The delineation between core components, internal, and external extensions (Eclipse runtime components, internal extensions-core projects, and plug-ins, respectively) is presented in the design of the community structure. At the innermost circle reside the developers of the Eclipse platform runtime. The inner circle consists of standard project developers and the outer circle of plug-in developers. Figure 7 shows the project community structure of Eclipse.

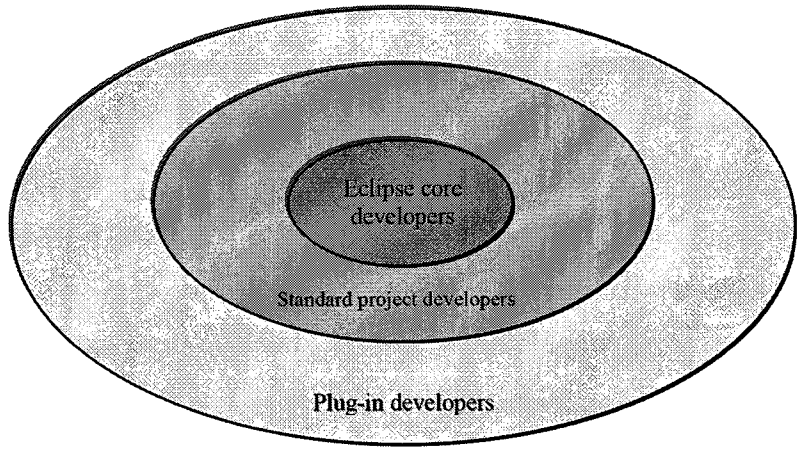

Figure 7. Eclipse community structure

The different handling of core projects and regular plug-ins illustrates a trade-off that the Eclipse Foundation as the platform owner makes. Core projects provide components for a variety of Eclipse "stacks". Depending on usage a different set of core projects is configured with the platform core, for example, the Web Tools platform for the development of enterprise Java applications, or the Eclipse Modeling Framework for the creation of model-based development tools. Careful filtering of

core project proposals and implementations ensures the quality of these widely used components. Regular plug-ins, on the other hand, are not exposed to such scrutiny, and can, therefore, evolve in unexpected ways that indicate new application domains for the Eclipse platform. The selection of these new opportunities is user-driven, hence ideas flow back to the platform owner.

Regulatory instruments

The Eclipse project has several instruments such as pricing, legal instruments, development process, and toolkits in order to control the quality of contribution for its core components and external extensions. The first is pricing. The Eclipse Foundation offers a membership program for external parties. The program offers different types for membership such as strategic, enterprise solution, or associate members and each type imply different levels of involvement in the evolution of the platform.

Legal instrument such as license and user agreements are employed in general for the purpose of protecting the platform contents but not the quality of the contents.

The Eclipse development process is another instrument for controlling the quality of the core components and internal extensions. The Eclipse development process is formally defined in governance documents (www.eclipse.org/org/documents) and updated on the Web site (www.eclipse.org/projects/dev_process). Each Eclipse project undergoes a strict process called "incubation". The incubation process is a regulatory

instrument that enables platform owners to filter the flow of ideas related to the core and internal extensions and assists platform owners to control the quality of extensions. Incubation is the first phase in the project life cycle. The project proposal is reviewed by the PMC or the EMO, after accepting the proposal the project goes through several validation cycles until a stable release is reached and all is supervised by the PMC or the EMO (Duenas, 2007). Figure 8 shows the stages of the Eclipse incubation process.

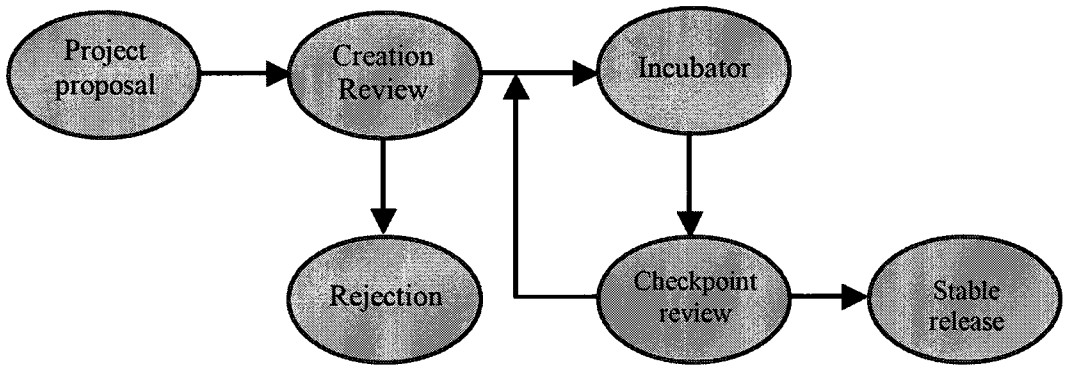

Figure 8. The Eclipse incubation process stages

Eclipse platform provide developers with toolkits to help controlling the quality of components developed by third-parties and these toolkits come in the form of plug-ins as well. There are toolkits that are part of the Eclipse project itself such as the Plug-in Development Environment (PDE), which is a comprehensive series of build and user interface creation tools, and tools for API maintenance. The Eclipse platform also has also toolkits developed by third-party and they reside in Eclipse Plugin Central

(http://www.eclipseplugincentral.com/), a repository for third-party plug-ins lists several tools for plug-in dependency analysis (such as eDepend) and code generation.

4.2 IBM WebSphere Application Server V6.1 and V7(WAS)

WebSphere is an IBM brand for software products that provide solutions for operating and integrating electronic business applications across multiple platforms. WebSphere products are based on Java technologies for both runtime components and application development toolkits (Sadtler et al., 2006). The purpose the WebSphere Application Server is to act like a middleware between back-end systems and clients. It provides different functionalities such as running business applications, developing and deploying web services, and a messaging engine (Keen et al., 2009).

Platform architecture

The IBM WebSphere Application Server V6.1 (WAS) was componentized in 2006 by adopting the OSGi architecture and the Eclipse extension framework that enabled the platform extensibility and provided the ability to avoid interdependencies and other problems by isolating the implementation code from the application code (Krill, 2009). The OSGi capabilities weren't fully exposed to developers in WAS V6.1 but for WAS V7 developers have more freedom to change the server configuration by bundling different components (Knoernschild, 2008). WebSphere Application Server consists of multiple containers and a stack of core services and infrastructure facilities that could be bundled in different configurations depending on the required features.

In general there are two main core components: a web container and an Enterprise Java Bean (EJB) container (Bernal, 2008).

The WebSphere extensions model follows the Eclipse extensions framework. The WebSphere extensibility mechanism enables the developers to implement external extensions at a predefined extension points in the WebSphere Application Server. WebSphere extensions could be an Eclipse tool or modules to contribute functionality such actions, task, or menu items. Figure 9 shows an overview of the WebSphere Server v6.x architecture.

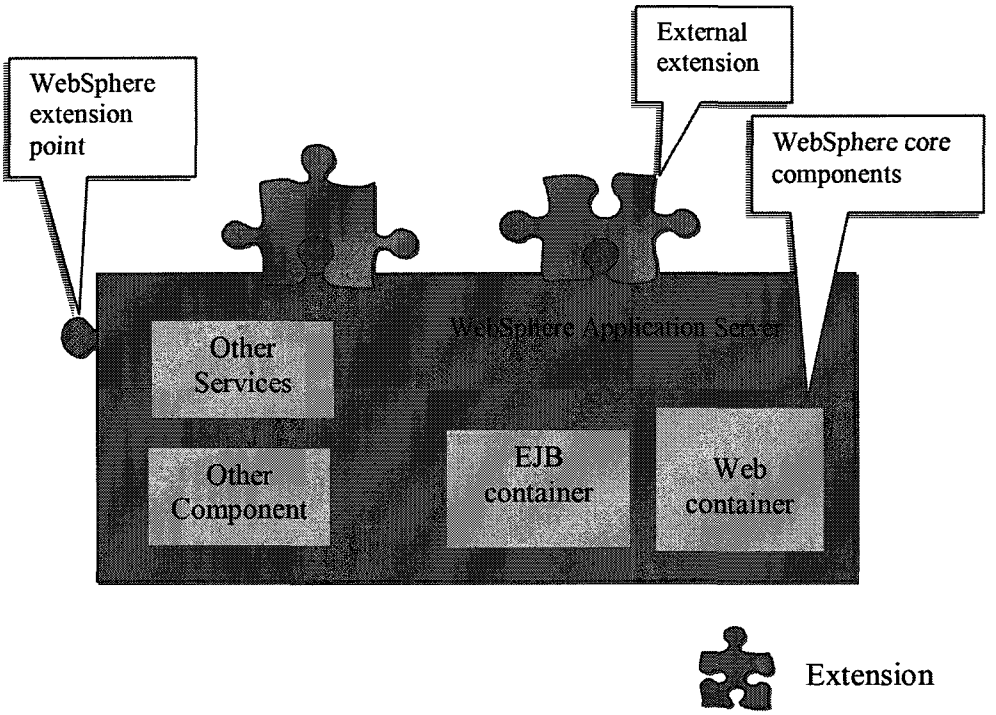

Figure 9. IBM WebSphere Application Server v6.x architecture overview

Collaboration Mode

The WebSphere programming extensions models enabled third-party developers to build extensions by defining extensions points in the WAS. Therefore, extensions developed are external to the core of the platform. The development decisions of the core component are completely strict to IBM developers but the extensions development is open to everyone in the community. Table 10 presents the collaboration modes for developing WebSphere extensions.

Table 10. Collaboration modes for IBM WebSphere Application Server extensions

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open			WebSphere extensions and plug-ins
	Closed	WebSphere internal components (e.g. Web container)		

Community structure

Figure 10 represents the community structure for the WebSphere Application Server. There are two levels of participating groups within the community. In the inner circle resides the IBM WebSphere team of developers, which is responsible for developing

the internal component and modules for the WebSphere Application Server. The external extensions developers reside on the outer circle.

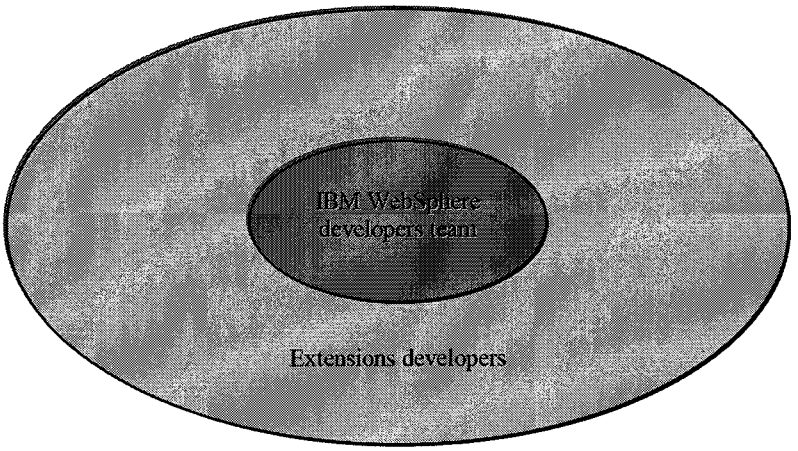

Figure 10. WebSphere Application Server community structure

Regulatory instruments

There is not any specific toolkit provided by IBM for developing and controlling the quality of WebSphere extensions. Most provided toolkits are for developing and deploying applications like the WebSphere Application Server Toolkit. Java troubleshooting toolkits are provided for developing and testing developed extensions and applications. Also IBM has a partnership program but not exclusively for developing extensions.

4.3 Mozilla Firefox Web Browser

The Firefox is an open source web browser released by the Mozilla Foundation. The Firefox browser project started as a branch of the Mozilla suite platform in 2003 and it kept evolving until the first major release in 2004 for Firefox 1.0. The Firefox web browser is considered one the best web browsers and has gained significant market share among other web browser like Internet Explorer and Safari in recent years (Yeow, 2005).

Platform architecture

The Firefox web browser is built on the Mozilla platform technology. The web browser itself consists of a set of packages known as chrome that describes the interfaces structure and style. The Firefox inherits the Mozilla platform extensible mechanism that allows third-party developers to write extensions to enhance its functionality by adding new features or enhancing the existing ones (Emura et al., 2007). Firefox allows developers to extend the web browser with new chrome packages or developers can replace the standard Firefox chrome altogether and radically reshape the web browser. An example of that is adapting Firefox for use on mobile devices (Spinellis & Gousios, 2009b). Packages that are related to the browser core functionalities or among its core feature are equivalent to the internal extensions such as the tabbed browsing extensions (Noll, 2007). Figure 11 presents an overview of Mozilla Firefox web browser architecture.

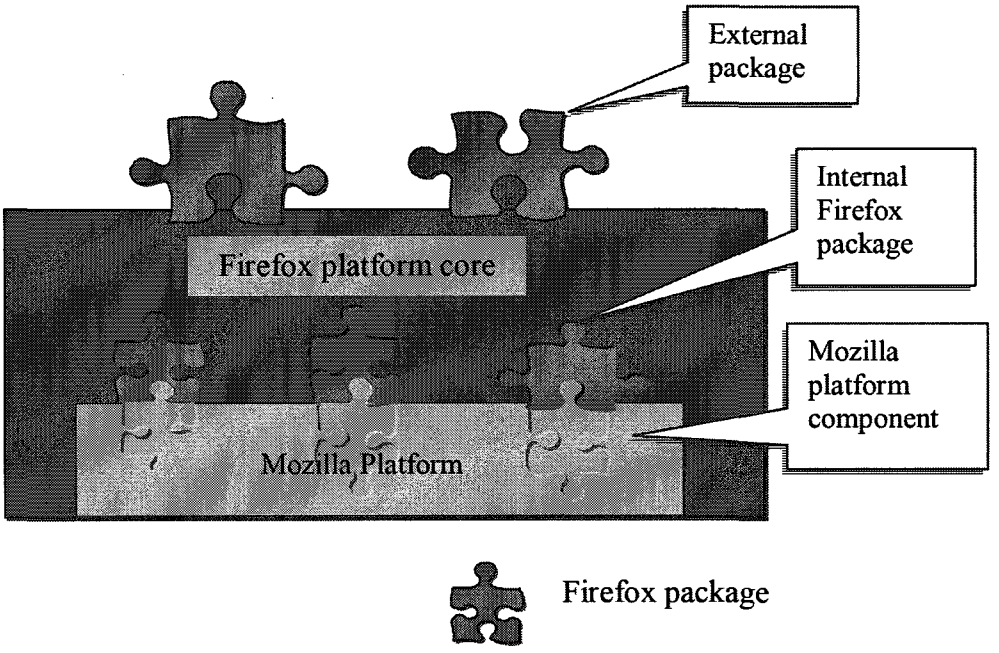

Figure 11. Mozilla Firefox platform architecture

Collaboration Mode

The Firefox project inherits its project management practices from a commercial product, Netscape, which provided the original source of the Mozilla project. Hence, the Firefox project is an open source project with a closed source flavor. It incorporates "traditional" software engineering practices such as formal requirements documents and usability studies, face-to-face meetings, paid developers, and a formal corporate infrastructure. The Mozilla framework defines guidelines for developers and sets levels of control that determine who can define requirements and commit code changes. Most Firefox extensions are created in an open manner, and defined and

selected by the community without involvement of the platform owner. Table 11 represents the collaboration modes for Firefox extensions. In general the Firefox extensions development process is an example for leveraging an open innovation network, and an example of flat governance. However, closer inspection shows that there are also examples of extensions which later became part of the Firefox core, which hence play the role of inner extensions. One example, identified by Noll (2007) is a Firefox feature for tabbed browsing. This feature was first introduced by the MultiZilla extension in 2000 and subsequently (in 2001) integrated as a core feature into the Firefox platform. The governance structure for the development of these extensions is a hybrid of hierarchical and flat, as the problem definition emerges from the community, but solutions are selected by the core development team.

Table 11. Collaboration modes for Firefox extensions

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open		Firefox internal extension	Firefox external extension
	Closed	Core components and Commercial extension		

While not very prominent, there are also examples of commercial extensions to Firefox. One example is the FullerScreen extension, which adds both a full screen and

slideshow mode for viewing web pages, and which is offered as a basic, free version and a paid-for, professional version. The governance structure for a commercial extension developed by a single company is hierarchical.

Community structure

Figure 12 shows the community structure for the Mozilla Firefox platform. It represents the different degrees of membership in the project as circles in an onion model. The innermost circle represents the developers who have commit rights to the Firefox core. The inner circle represents the developers whose extensions are accepted as part of the Firefox core. To represent this special status of such extensions, we consider them on the same level as internal extensions. However, since they are not formally labeled as such in the Firefox architecture, we show the inner circle as a dashed line in the diagram of the Firefox community structure in Figure 12. The outer circle represents the developers of all other that is, external Firefox extensions.

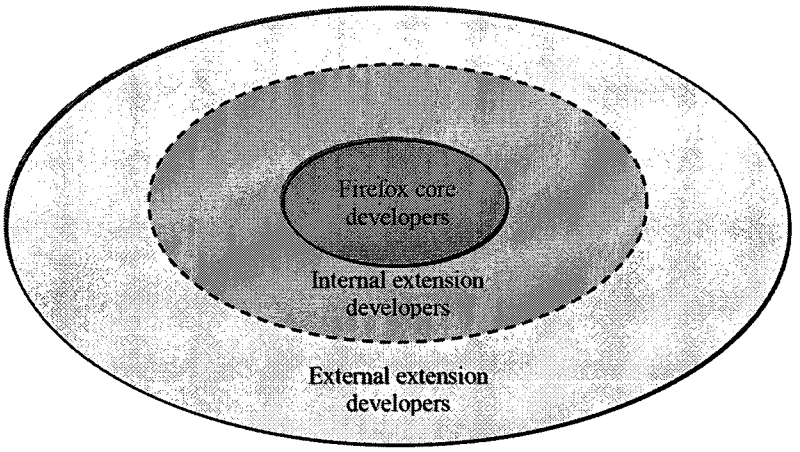

Figure 12. Mozilla Firefox community structure

Regulatory instruments

To ensure the quality of extensions, the Mozilla project offers a toolkit consisting of several tools to test the performance and quality of Firefox extensions. These include a web-based test case management tool, Litmus, for performing smoke tests and creating test cases, and the Bugzilla bug tracking system for tracking feature requests and errors. For example, a Litmus test suite to assess the accessibility of an extension is provided by the Mozilla project. The Tinderbox As Eclipse provides the Plugin Development Environment, Mozilla provides the Tinderbox for as a tool to provide continuous integration capability, it is an investigative tool to help the developers to manage and code.

A number of Mozilla development extensions are also provided by third parties to aid in the development of Firefox extensions. These third-party toolkits include editors and debuggers (such as Firebug), inspectors (such as Chrome List), and packaging tools (such as Extension Developer). There are also third-party tools (wizards) for the creation of the initial code skeleton of an extension.

Alongside with development toolkits, Mozilla provides the community with the "Sandbox Review System" process to control quality of developed extensions. In this process advanced users test the new add-ons before they are reviewed for public use. This process helps to filter out the ideas and control the quality of contributed extensions. Figure 13 illustrates the stages for the Sandbox process.

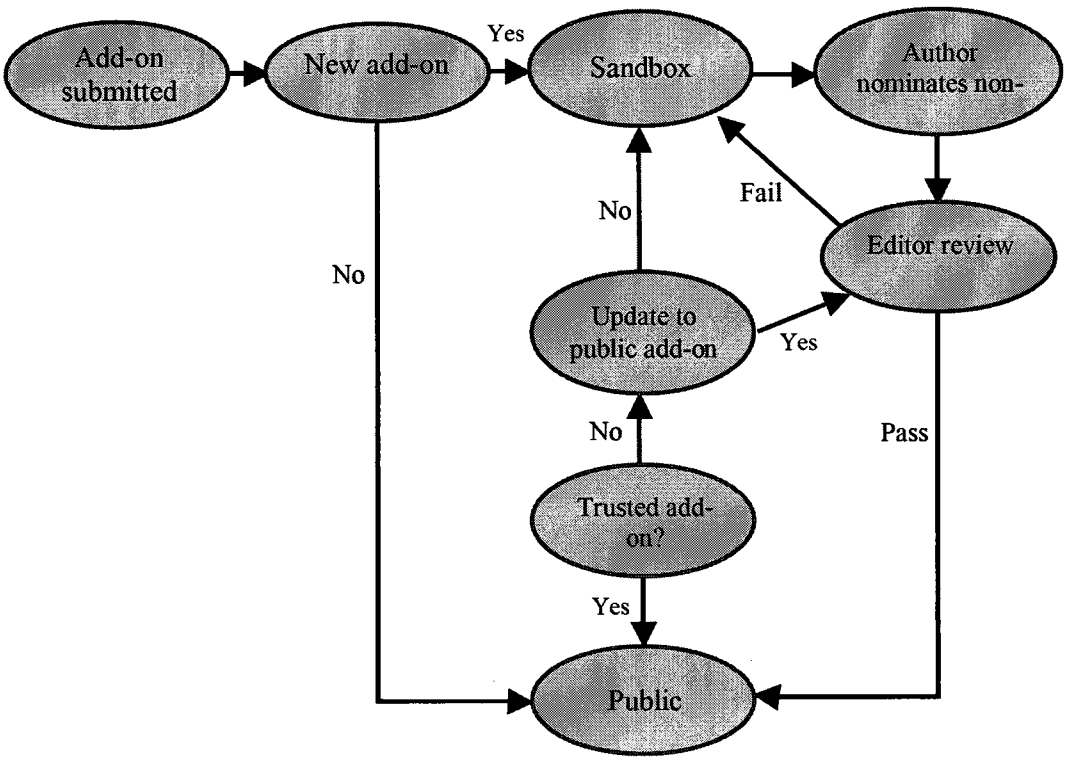

Figure 13. The Mozilla Sandbox Review System

4.4 Spring framework (open source edition)

The SpringSource platform (known as Spring framework) is a Java and .NET framework for building enterprise application. It is an open source project sponsored by SpringSource.org. The Spring platform was published in 2002 by Rod Johnson and described in his book Expert One-on-One: J2EE Design and Development and the first official 1.0 release was in March 2004 (Walls & Breidenbach, 2007).

Platform architecture

Spring platform is composed of a number of well defined modules that gives the freedom to pick needed modules to develop applications. The Spring modularity enables extending the platform through adding new modules or enhancing existing ones. The Spring platform consists of sets of core modules (such as Java Management Extensions - JMX, Java EE Connector API - JCA) and extension modules that sits on top of the Spring core container and each one module is developed and maintained by a SpringSource.org project (Hemrajani, 2006a; Machacek et al., 2008). Figure 14 shows the architecture for Spring platform.

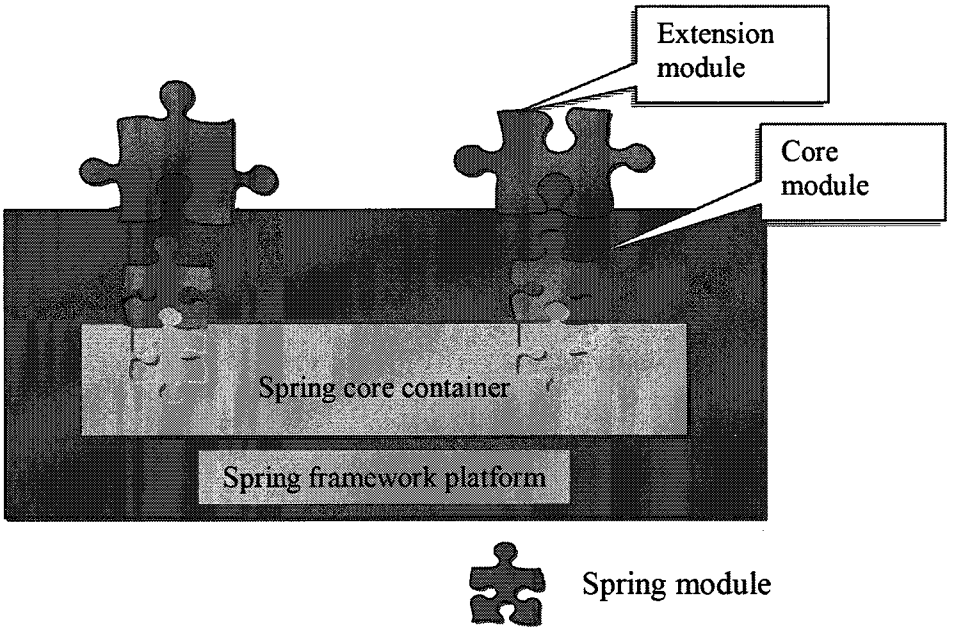

Figure 14. SpringSource platform architecture

The Spring core modules are equivalent to the internal extensions because they are part of the framework architecture. Beside the core modules there are also other modules that are integrated with the Spring platform like the Spring Web Flow and other that can be considered as external extensions. In 2009 the SpringSouce.org established the Spring extensions project to organize the process of extending the Spring external modules or the extensions module. The extensions project consists of subprojects that each represents a Spring extension module. These developed extensions could become an external module at the end of its life cycle.

Collaboration Mode

The documentation of Spring extensions development process is well described on their website (http://www.springsource.org/extensions/proposing). The process of developing SpringSource core and extension modules is tightly governed by the SpringSource.org and every Spring module and extension is endorsed by SprinSource.org. Each extensions project has an internal SpringSource sponsor to monitor the project's development process and ensure its quality meets SpringSource standards.

Beside the free/open source extensions, there is a set of commercial extensions for the Spring platform that we are going to discuss it separately in section 4.5.

By enforcing strict rules to govern the community contribution, SpringSource.org aims to control the quality of extensions and direct the flow of ideas towards its goals to provide a high quality product to compete with other products in the market. Table 12 shows the mode for SpringSource collaboration.

Table 12. Collaboration modes for SpringSource.org extensions

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open	Spring core and extension modules		
	Closed	Spring core container and commercial modules		

Community structure

Figure 15 shows the structure of the SpringSource.org community for developing external modules and extension modules. The Spring developers team resides in the inner circle. The responsibilities of the core team members are: i) develop the Spring core components - the Spring core container, and ii) supervise the development process of every Spring module (core modules and extensions modules). For every SpringSource project there is an internal SpringSource supervisor to oversee the

project progress and ensure the quality of the outcome. Therefore, the core modules developers and the extensions modules developers are governed by the same rules and under the same level of control. The outer circle represents the community of developers for the core modules and extensions.

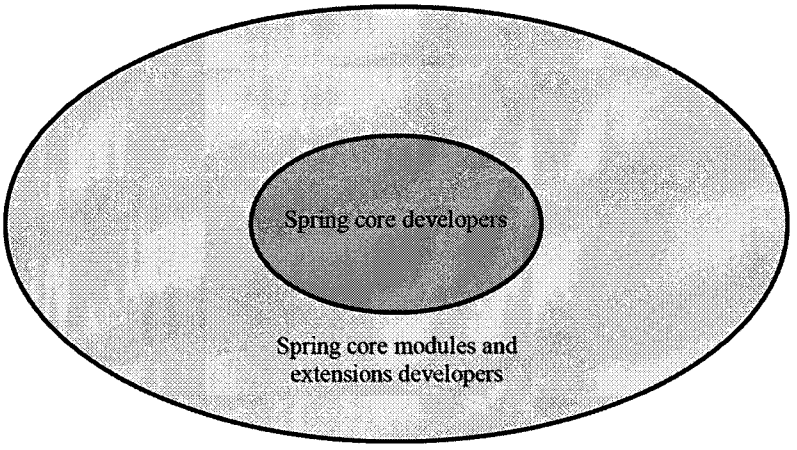

Figure 15. SpringSource.org community structure

Regulatory instruments

The SpringSource.org offers a general purpose toolkit for developing Spring modules, extensions, and applications and it is not an exclusive toolkit for developing extensions. The SpringSource Tool Suite (STS) is an extensive development environment for developing and testing Spring modules. Since April 2009 STS toolkit became available for free and this toolkit has many features like OSGi development

tools or Dynamic Modules Server Integration that help developing Spring components and applications (Dupuis, 2009).

As toolkits is one way to control the quality of developed components, SpringSource established a process exclusively for developing extensions and nurture them until they graduate to a full module within the Spring platform. The extensions lifecycle process adopts the same concept of the Eclipse incubation process. The extension lifecycle consists of three stages: proposal, incubation, and live or archived (as shown in Figure 16).

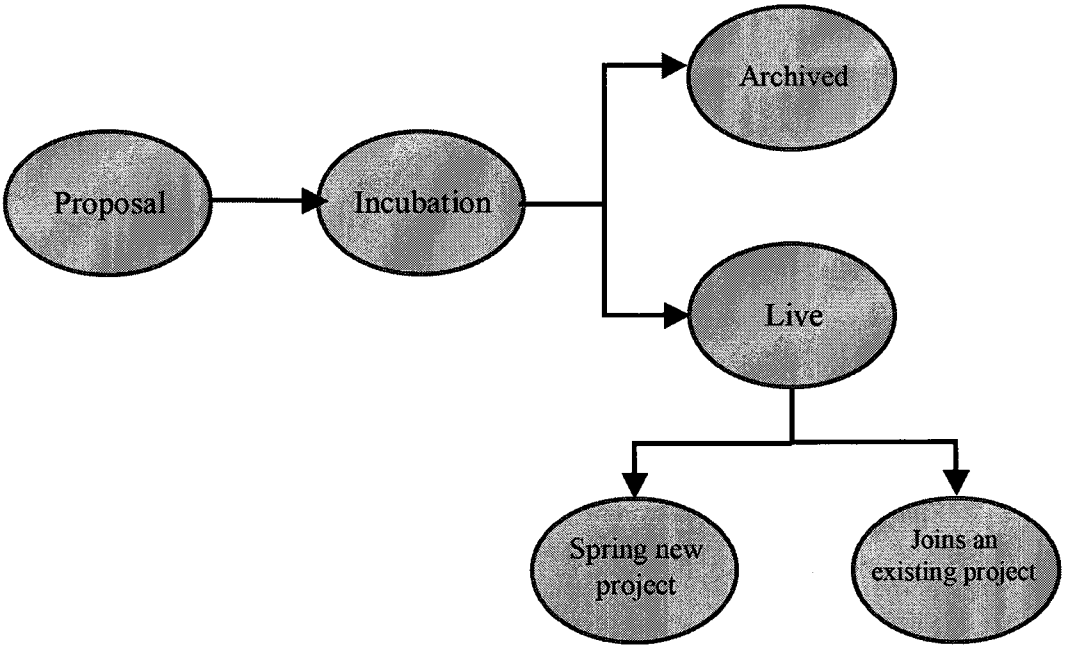

Figure 16. The SpringSource extension lifecycle

The first stage is the extension proposal which leads to incubation stage if accepted. An extension project stays in the incubator stage until it reaches a stable level of feature completeness and maturity. An extension project will be archived if it does not have enough resource such as developers or it becomes redundant to a core project. Otherwise, when the extension is proven to be mature and stable enough, it will become a live project where it will either evolve to spin a new project or to join an existing project.

4.5 Spring Framework (enterprise edition)

In July 2008, SpringSource released the Spring platform under a duel licensing model and introduced its first commercial product, "SpringSource Enterprise", and the springsource.com represent the SpringSource Enterprise products and community.[3] The SpringSource Enterprise includes a certified version of the open source Spring platform and other Spring module like Spring Security and Spring Webflow (Romahn, 2008).

Platform architecture

The SpringSource Enterprise platform is based on the same architecture of the open source platform but all the components are tested and certified by SpringSource.

---

[3] SpringSource Announces General Availability of SpringSource Enterprise. (http://www.springsource.com/newsevents/springsource-announces-general-availabi). Retrieved on August 30, 2009.

Added to the open source platform extensions project, the SpringSource provides the "Exchange" repository (http://www.springsource.com/exchange) to publish certified commercial modules and applications that are developed by third parties.

Collaboration Mode

The SpringSource enterprise inherits the SpringSource.org developer community from the non-commercial core models and extension modules. In the case of the enterprise edition, the certified commercial core and extension modules are developed by SpringSource developers and certified parties. These modules are published at the "exchange" repository. Table 13 shows the collaboration mode for the SpringSource commercial version.

Table 13. Collaboration mode for the commercial Spring framework

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open	Internal extension modules		
	Closed	Spring commercial core and extension modules		

Community structure

Figure 17 shows the structure of the SpringSource.com community for developing core and extension modules. The Spring core team resides in the inner circle as it is has the same privileges as in the case of the open source project but it also supervises the development process for the commercial extensions that are developed by third parties. The SpringSource.com offers a partnership program to attract members to its developer network for commercial extensions. The commercial modules developers lie on the outer circle, the same as in the open source model. The difference between SpringSource.org and SpringSource.com community structures is network openness. In the case of SpringSource.org the participation is open to everybody in the community but the process is regulated. For SpringSourc.com, the participation for the commercial extensions is limited to the members of SpringSource partners club.

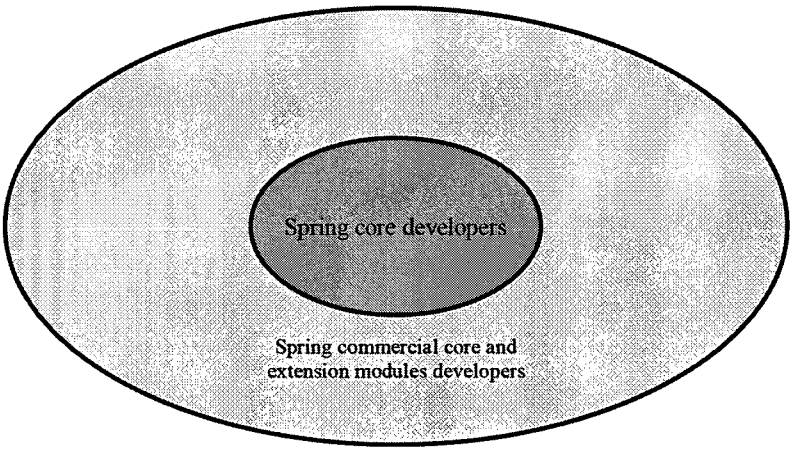

Figure 17. SpringSource Enterprise community structure

Regulatory instruments

In the case of the SpringSource enterprise, there are two instruments used to control the flow of ideas in the network of the enterprise edition and control the quality. First SpringSource offers a partnership program and certification program for developing Spring enterprise modules and applications.

Second, the SpringSource enterprise offers the enterprise version of the SpringSource Toolkit Suit (STS) for the enterprise modules and application developers.

4.6 Apache HTTP Server

Apache HTTP server is open source web server and it is one of the Apache Software Foundation many projects. The Apache Software Foundation (ASF) provides the organizational, legal, and financial support for the Apache web server project and the other Apache projects such as Apache Ant, Apache Tomcat and many others. In general the Apache projects attract high number of users because of the products' high quality and the Apache license that makes it easy to deploy Apache products (Stein & MacEachern, 1999).

Platform architecture

The Apache HTTP Server consists of a small core (the Apache Portable Runtime) and number of modules that can be compiled into the server statically or loaded dynamically at runtime. The Apache Portable Runtime (APR) provide a cross-

platform operating system layer and utilities for running the Apache modules. The Apache HTP server modular architecture enables the users to add feature they need and integrate them with the system (Ridruejo & Kallen, 2002; Kew, 2007). The Apache Server has two types of modules: set of core modules[4]that comes with the standard distribution and a set of open source and commercial modules that are developed by third parties. The core and third-party modules are equivalent to the internal and external modules respectively. Figure 18 represent a simplified architecture for the Apache HTTP Server.

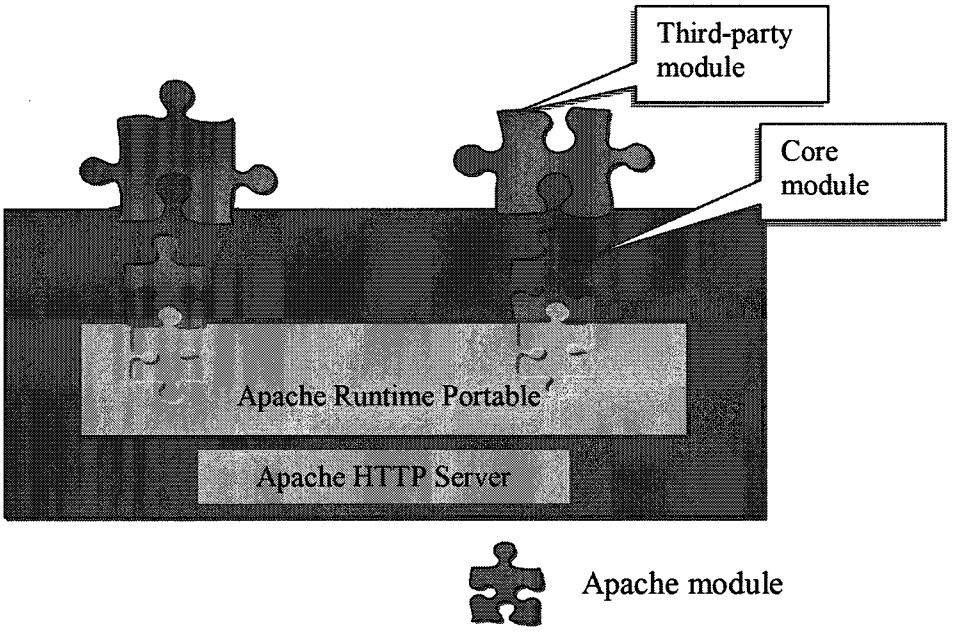

Figure 18. Apache HTTP Server architecture

[4] Apache HTTP Server core modules list could be found at (http://httpd.apache.org/docs/2.0/mod/).

In 2002 the Apache HTTP Server 2 was released and it introduced new levels for extending the server. Apace HTTP Server 2 can be extended in three ways (Lopez & Blanco, 2006):

- Multi Processing Modules (MPMs): Allow you to change the way Apache serves requests and improve the performance and scalability of the server.

- Filtering Modules: Provide a way for modules to process the content provided by other modules.

- Protocol Modules: The protocol layer has been abstracted, so it is possible for Apache to serve content using other protocols, such as FTP

Collaboration Mode

Table 14 shows the collaboration mode for the Apache HTTP Server project. The Apache HTTP Server project is a meritocracy, which means it is governed by merit. A developer has to earn the right to commit to the project by showing an adequate understanding of the ASF processes and procedures as well as contributing a high quality contents (e.g. code patches, documents, etc). These rules are applied for all the Apache server core modules such as core, mpm_common and other modules that can be found at (http://httpd.apache.org/docs/2.0/mod/). The above process helped to maintain the high quality of the Apache server and its components (Stein & MacEachern, 1999).

Table 14. Collaboration mode for the Apache HTTP Server

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open		Apache sever modules	Third-party modules
	Closed	Apache core and commercial modules		

The contribution to the external modules repository has no specific process or restrictions for publishing contents aside from a simple registration process and approving the module. The repository contains open source and commercial models as well as commercial web servers based on the Apache server. In the case of the commercial modules we find that the development process is restricted to the sponsor company like in the case of BEA and WebLogic (Ridruejo & Kallen, 2002). In the case of the external extensions, having a flat governance structure allowed a high flow of ideas, which was reflected on the number of published modules, but there is no evidence that such modules are at high quality.

Community structure

There are three levels of projects in the Apache community, Apache Top-Level-Project (TLP), Apache Projects, and Apache Subprojects. A TLP is at the top of the pyramid for the Apache projects and the ASF board decides what project could join.

The second level is the Apache projects where individuals desire to add or implement new features and accepting this type of projects is subject for voting. Each project has an appointed group called Project Management Committees (PMC) to oversee the development.

The last type is the Subprojects and the one of our interest is the Apache Modules (http://modules.apache.org/). The Apache modules project is a repository for third-party developers to register, publish their modules, and to share them with the Apache community.

In the case of the Apache HTTP Server project, modules developers can be divided into two groups: the core modules developers and the external modules developers. The core modules projects are under the same rules of an Apache standard project. The external modules are considered a subproject of the Apaches HTTP Server project and individuals are free to register and publish the contents on the modules repository. Figure 19 shows the structure for the developer community of the Apache HTTP Server.

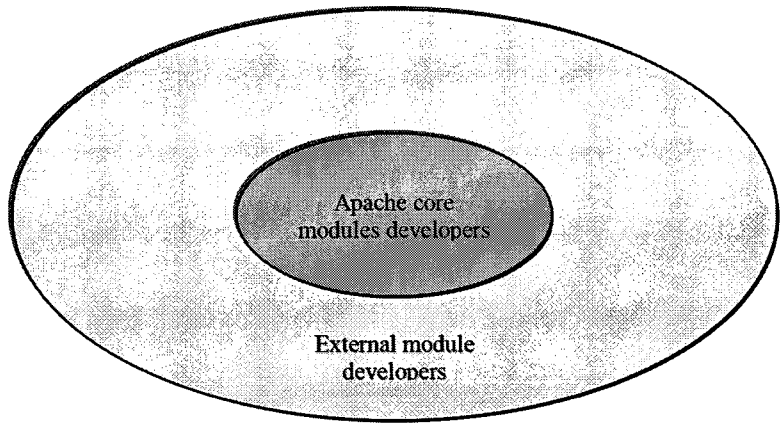

Figure 19. Apache community structure

Regulatory instruments

The Apache HTTP Server project does not provide toolkits exclusively for developing and controlling the quality of the server modules. There are subprojects within the Apache HTTP Server for developing tools for testing and developing applications such as the Flood project, Perl Framework, and Apache Test. The Apache HTTP Server uses the Apache Bugzilla reporting bugs, fixes, and requests for new features.

The ASF established the project "Incubator" that has the same concept of the Eclipse incubator. The incubator is the entry point for new project to the ASF projects and codebases. The incubation process can be considered a method to filter the flow of ideas as well as to ensure that the new projects are following the ASF guidelines for coding and quality. The Apache incubation process and policy details are described at the incubator policy page (http://incubator.apache.org/incubation/Incubation_Policy.html). Figure 20 describes

the stages of the Apache incubator. The incubation process covers the establishment of a candidate, acceptance (or rejection) of a candidate leading to the potential establishment of a Podling and associated incubation process, which ultimately leads to the establishment or a new Apache Top-Level-Project (TLP) or sub-project within an existing Apache Project.

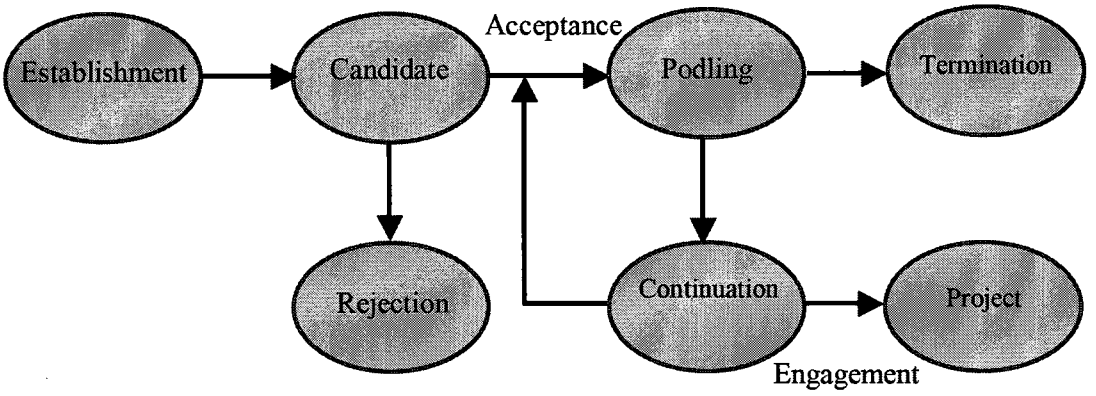

Figure 20. Apache incubator project stages

4.7 IBM HTTP Server

The IBM HTTP Server is a pre-packaged Web server based on the Apache Web server. IBM has supported and enhanced Apache server versions since 1998. IBM offers the IBM HTTP Server at no cost but it offers support with the purchase of the IBM WebSphere Application Server. The IBM HTTP Server includes several enhancements and additional functions that are not available in the Apache Web server. This relationship between IBM and the ASF comes with mutual benefits for both organizations as IBM has the access to the ASF great pool of innovation, and

ASF gains from the contributions of IBM engineers and developers (Johner, 1999; Chen et al., 2006).

Platform architecture

The IBM HTTP Server release is an Apache HTTP Server integrated with IBM developed modules. IBM implemented some limited changes to Apache HTTP Server modules themselves and the Apache Runtime Portable (APR) when creating IBM HTTP Server (Johner, 1999). There are two main areas for changes implemented by IBM:

1. Bug fixes: The IBM development team selectively backports fixes from the Apache HTTP Server and APR development source code repositories IBM HTTP Server.

2. Changes to support IBM modules: IBM enhances the Apache server functionality by adding a graphical installer module, adding an integrated Secure Sockets Layer (SSL), enhance the performance by improving the memory management modules, LDAP integration. Therefore, IBM maintains changes to these modules in order to support specific IBM requirements.

Since the IBM web server is based on the Apache web server, users can integrate other Apache modules but IBM provides support only for modules that come with the IBM HTP release.

Figure 21 shows us the IBM HTTP Server platform architecture. The enhanced Apache core and ARP modules reside at the core of the IBM web server platform. IBM extends the Apache web server functionalities by adding IBM developed internal modules such as the SSL modules or the LDAP modules (Chen, 2006). IBM allows installing any other third-party modules such as Apache modules that can be found on the Apache modules registry. The third-party modules are considered external to the IBM web server and IBM does not support them.

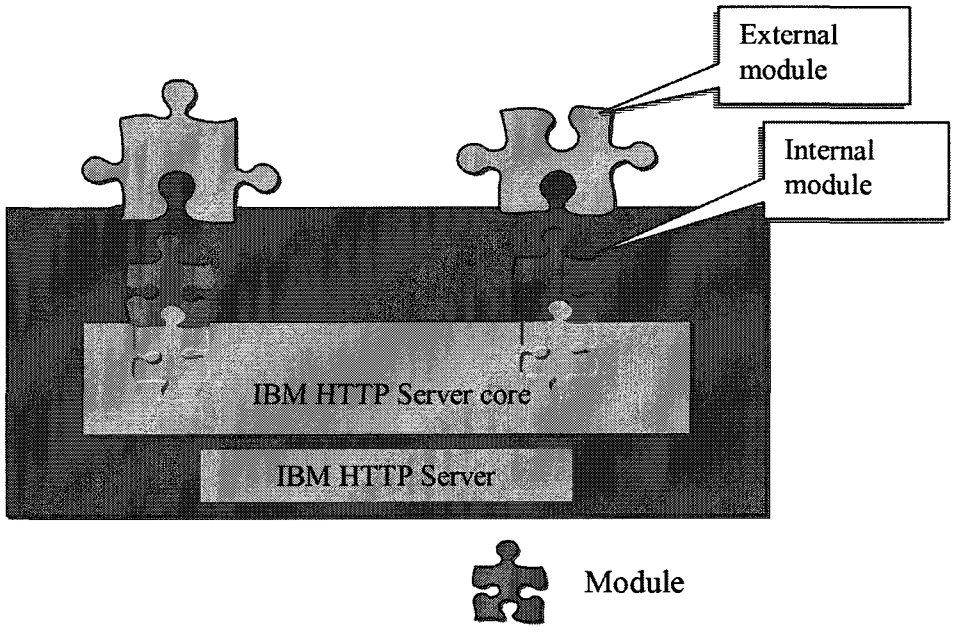

Figure 21. IBM HTTP Server architecture

Collaboration Mode

The development of the IBM HTTP Server components (core and internal extensions) is limited to IBM engineers and developers. So the developer network is closed for

core and internal extensions, because it is limited to IBM personnel only. On the other hand, external extensions (i.e. third-party modules) developer network is open since IBM has no restrictions for such process and provides no support for quality control for such extensions.

With this approach IBM is controlling the quality of components that seen critical for the web server performance and allows a greater flow of ideas to its network by enabling external extensions integration.

Table 15 represents the collaboration mode for the IBM HTTP server for developing extensions.

Table 15. Collaboration mode for the IBM HTTP Server

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open			Third-party modules
	Closed	IBM internal and core modules		

Community structure

IBM limits the rights of development for the IBM added modules and the enhanced Apache web server to its own team of engineers and developers. While developing external extensions is open to third- parties without any restrictions.

Figure 22 represent the developer community structure for the IBM HTTP server extensions. The community can be segregated into two groups. First is the IBM developers group that builds and develops the IBM web server components including the enhanced Apache modules. The second group is the third-parties that develop external extensions to be integrated with the IBM server such as Apache modules.

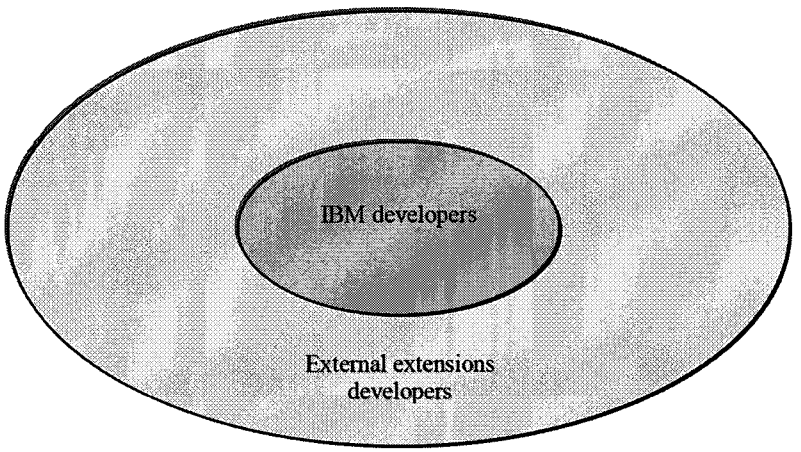

Figure 22. IBM HTTP Server community structure

Regulatory instruments

IBM does not provide an exclusive development toolkit, sandbox, or membership program for developing the web server components or for Apache web server

modules. Although IBM provides a range of toolkits and training programs that support developing applications run on top of its products. An example of toolkits is the WebSphere Studio Application Developer or the WebSphere SDK for Web Services.

4.8 OpenOffice.org

The OpenOffice.org is an open source project to provide an office application suite and it is based on Sun Microsystems technology of StarOffice productivity suite. Sun Microsystems is one of the major sponsors of the office suite and the OpenOffice.org council committee is the governing body for the OpenOffice.org community (Shibuya & Tamai, 2009).

Platform architecture

The OpenOffic.org platform consists of a collection of extension packages that allows small deployments (Schmidt, 2006). The OpenOffice.org extension packages are known as the Universal Network Objects (Uno) components. The Uno Runtime Environment (URE) is a set of common libraries to support the Uno components and make it available to run the Unos independently (Tenhumberg, 2005; Schmidt, 2006).

There are two levels for developing OpenOffice.org components: product level and extensions level. The product level represent the group of projects are related to developing OpenOffice.org core functionalities such as word processing, security,

database access, and Uno development kit[5]. The components that are developed at the product level considered internal to the platform. The extensions level includes the group of projects are related to developing components that extends the OpenOffice.org and extensions' management components[6]. The components developed at the extensions level are external to the platform and it includes add-ons, plug-ins deployed by Uno packages. Figure 23 represent the OpenOfice.org platform architecture.

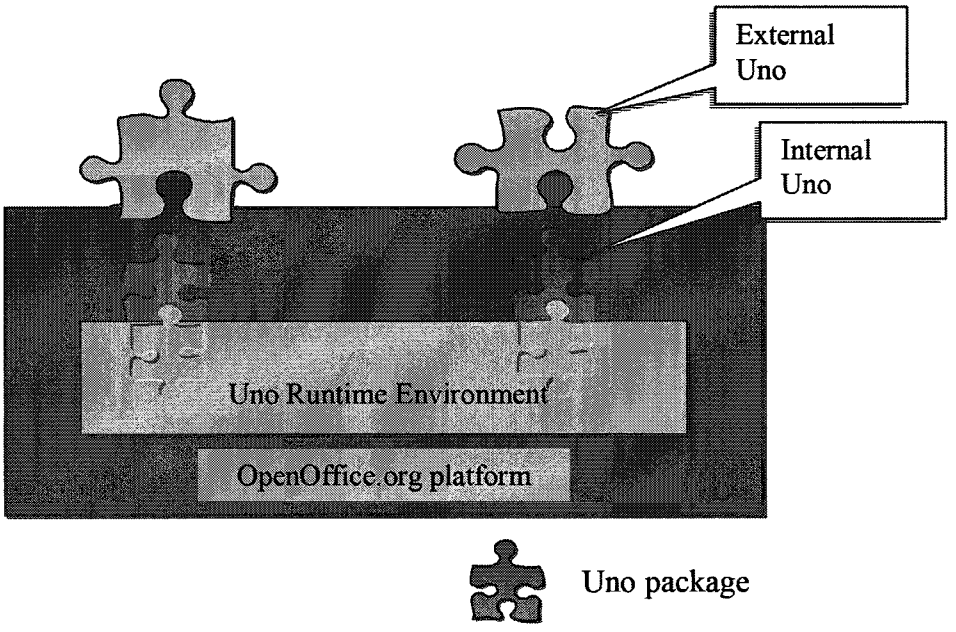

Figure 23. OpenOffice.org platform architecture

[5] The complete list can be found at (http://projects.openoffice.org/index.html#components).
[6] The list for extensions project can be found at (http://projects.openoffice.org/index.html#extending).

Collaboration Mode

The OpenOfice.org community is governed by the OpenOffice.org Community Council. The council is a group of people who are active contributors to the OpenOffice.org project and the council must include one Sun Microsystems representative who is appointed by Sun. The council major duties are related to strategic planning, resource allocation, managing the relationship between the sponsors and the OpenOffice.org communities. The duties and powers of the council are described in details in the Community Council Charter[7]. The council is also responsible for accepting new projects and pass recommendations for project lead on the direction of the projects.

Each OpenOffice.org project has a project lead and co-lead to supervise the development process and manage the community of the project and report to the council. So the council acts first as a filter for the flow of ideas in the OpenOfice.org community and second as a mediator between the community and Sun Microsystems. However, the OpenOffice.org is a meritocracy and it incorporates voting as a mechanism for decision making (Shibuya & Tamai, 2009).

As mentioned in the previous section, OepnOffice.org project has two levels of development: product development for internal extensions and extensions

---

[7] Community Council Charter, (http://council.openoffice.org/councilcharter12.html), retrieved at June 18, 2009.

development for external extensions. The product development level includes the public projects (or the core projects), and these projects include three project types: accepted projects, native language projects, and the incubator projects.

The accepted and native language projects are the projects related to developing the core components of the OpenOffice.org or the internal extensions for the platform. Although, OpenOffice.org is open for outside developers to participate but the process for joining a public project or proposing a project is a tightly controlled process and it is well documents on the OpenOffice.org website at (http://www.openoffice.org/dev_docs/guidelines.html) and (http://www.openoffice.org/about_us/protocols_proposing.html#proposingw).

The "incubator" projects are the newly proposed projects and that process is similar to the Apache incubator but only OpenOffice.org registered members have the right to propose new projects. Proposed projects have to get enough community votes to be created and join the incubator projects. The incubator has fewer restrictions to join a project because it is considered a testing ground for new ideas and provides opportunity for new members to join the community. Incubator projects fate is either promoted to join the accepted projects or die for lack of resources or other reasons.

The extension level includes the extensions project, which was created in 2006 to help produce, organize and deploy OpenOffice.org extensions (Suàrez-Potts, 2006). For the extensions project, the process to submit an extension is very open to third parties as

anyone can submit extensions as long as they are registered. There are certain guidelines to follow in submitting extensions regarding the format and packaging. OpenOffice.org provides a repository for third-party extensions (http://extensions.services.openoffice.org/) where open source and commercial extensions are published. The commercial extensions case, we found that the parties developing such extensions are limiting the development process to their own personnel; therefore, their network is closed. Table 16 represents the collaboration mode for the OpenOffice.org community.

Table 16. Collaboration modes for OpenOffice.org extensions

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open		OpenOffice.org internal extensions	OpenOffice.org extensions
	Closed	OpenOffice.org commercial extensions		

Community structure

Figure 24 represent the OpenOffice.org community structure for developing internal and external extensions. The pubic projects are related to developing OpenOffice.org core and internal extensions; therefore, there are restrict rules to join the development

teams and ensure the quality of the final products are up to the Sun Microsystems standards. The incubator projects though have fewer restrictions than other types of the public projects because they are considered a space for new developers to test and develop ideas. The developers for both accepted and native language projects reside in the innermost circle and the incubator developers reside in the inner circle in figure 24.

As for the extensions projects developers, they reside in the outer circle because the development process has no restrictions for participating in developing extensions projects.

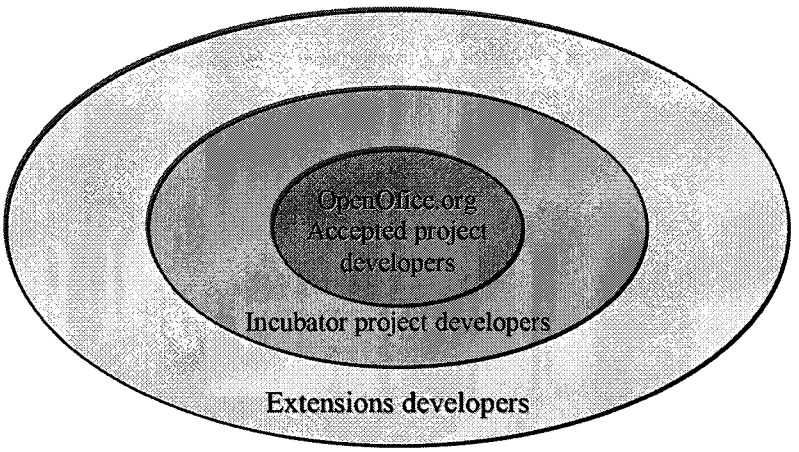

Figure 24. OpenOffice.org community structure

Regulatory instruments

OpenOffice.org utilizes different instruments to control the quality of developed extensions and control the flow of ideas. OpenOffice.org provides a set of toolkits for

developing and testing extensions as well as third-party developed toolkits to make developers able to test the different components for quality and performance. The main toolkit available for public is the Test-Case Management tool (TCM), this tool is an OpenOffice.org specific QA project to test new and existing OpenOffice.org components including extensions. To deploy extension, OpenOffice.org provides an extension manger and the extensions framework to help extension developers by providing methods for extension development translation and extension introspection. Beside the toolkits provided by OpenOffice.org, third-party toolkits like: XRay, pyXRAY, and pyUNO are available for the community.

OpenOffice.org also has established a quality assurance procedure for testing new feature components where a quality assurance engineer is assigned to oversee the development and test processes of the new feature components.

The incubator project is one more form of instruments used to control the quality of developed extensions and filter the flow of ideas coming to the platform network. Incubator projects follow the process represented in figure 25.

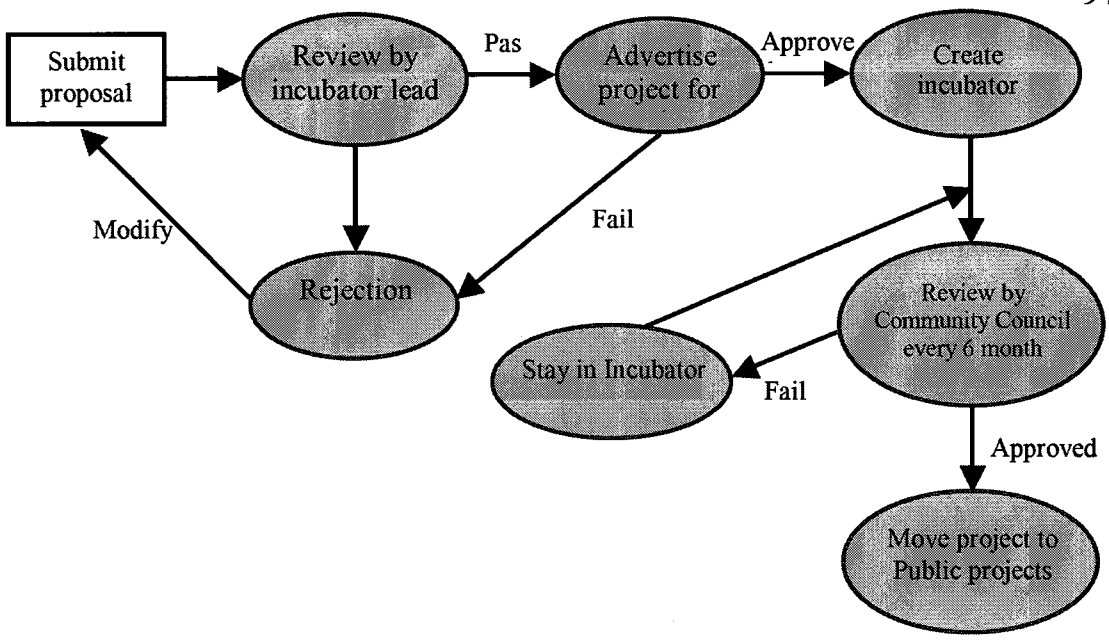

Figure 25. OpenOffice.org incubator project stages

4.9 StarOffice

StarOffice is the proprietary office suite package provided by Sun Microsystems. The StarOffice suite was developed first by StarDivision and acquired by Sun in August 1999. In year 2000, Sun released StarOffice code under the GNU Lesser General Public License (LGPL) open source license to create the free, open source office suite "OpenOffice.org".

Platform architecture

The StarOffice architecture is based on the OpenOffice.org platform. Sun Microsystems integrates the OpenOffice.org with its own proprietary code and third-

party code then releases it under a commercial license. The StarOffice has its own proprietary extensions as well as OpenOffice.org extensions in the form of add-ons and plug-in (Sun Microsystems, 2008).

Since StarOffice 8, it was made easy to write extensions like add-ons and plug-ins for StarOffice and in StarOffice 9 included a modified extension manager and extension framework to help developers write extensions for both OpenOffice.org and StarOffice (Sun Microsystems, 2008).

Therefore, the StarOffice platform consists of the OpenOfffice.org codebase integrated with Sun's proprietary components. The proprietary components are equivalent to the internal extensions because they are essential to the StarOffice functionality. The OpenOfice.org extensions and other StarOffice third-party extensions are equivalent to the external extensions and they take the form of Add-ons and plug-ins deployed in an Uno package. As shown in figure 26.

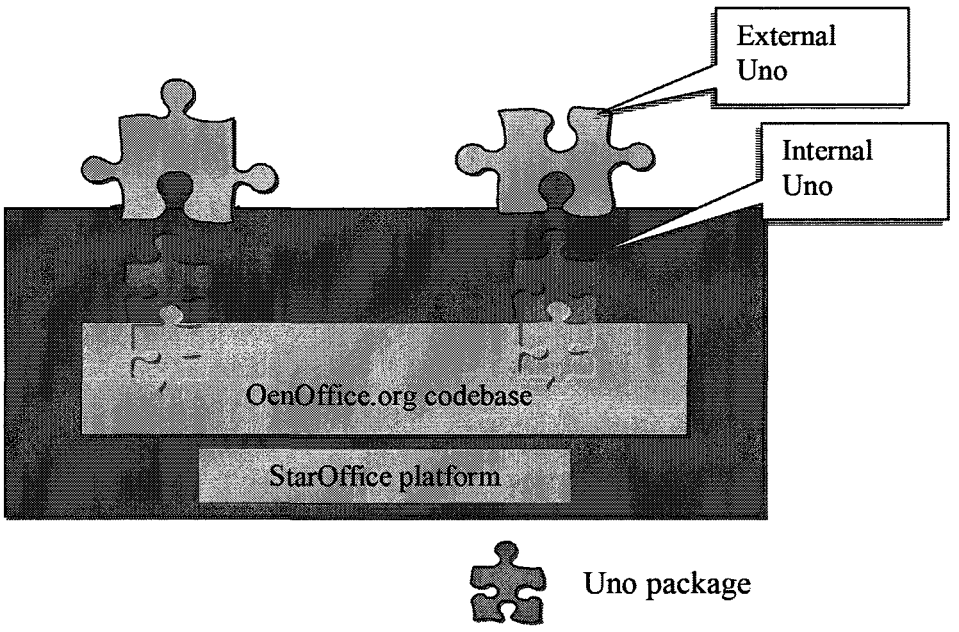

Figure 26. StarOffice platform architecture

Collaboration Mode

StarOffice is basically a snapshot of OpenOffice.org and bundled with Sun's proprietary code. So Sun developers carry out the task of developing StarOffice internal to control the direction of the development process and the quality at the same time. Therefore, the governance structure for developing the platform internal extensions is hierarchical because of the tight control of Sun Microsystems.

Since StarOffice external extensions includes the OpenOffice.org extensions, then it inherits the collaboration mode for these extensions. Also the StarOffice extensions in

general are open to the community and no restrictions applied. Table 17 shows the collaboration modes for the StarOffice extensions.

Table 17. Collaboration modes for StarOffice extensions

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open			OpenOffice.org and StarOffice extensions
	Closed	StarOffice internal extensions.		

Community structure

Sun limits the internal extensions development to its own StarOffice development team. But for external extensions, the development process inherits the OpenOffice.org process for developing external extensions. So developing StarOffice external extensions is open to the community and the same OpenOffice.org rules apply for StarOffice.

Figure 27 shows the structure of the community for developing StarOffice extensions. In the inner circle reside the Sun Microsystems developers as the development process falls under strict rules to control the quality of the product and high influence by Sun.

The outer circle represents the third-party and OpenOffice.org developers since Sun does not control the development process or the quality of the developed extensions.

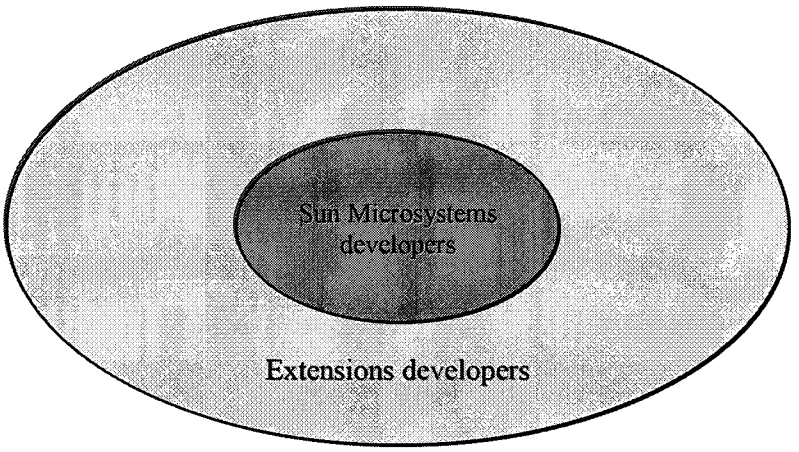

Figure 27. StarOffice community structure

Regulatory instruments

Sun controls the quality of developing internal extensions by limiting the development process to its own developers. For the external extensions, the OpenOffice.org toolkits are used for developing StarOffice extensions. Also since StarOffice 8, the office suite included a new framework for integrating external extensions and enhanced extensions manager.

4.10 JBoss Application Server

The JBoss application server started as a Java 2 Enterprise Edition (J2EE) compliant open-source application server and the latest edition is based on Java 5 Enterprise Edition. The application server provides component models-standards that developers can use to develop components and a standard deployment model for deploying these components. The server also provides a set of services for the running components (Jamae & Johnson, 2009).

The JBoss application server was founded by a group of developers led by Marc Fleury and in the year 2001, Fleury and company incorporated as the JBoss Group, LLC, and started offering developer support services in 2002 (Watson et al., 2005) .

The name of the open source project was originally EJBoss (Enterprise Java Beans Open Source Software). Sun did not like the use of their EJB trademark, so the "E" was dropped from the project name, making it JBoss. Red Hat, Inc., bought JBoss, Inc., in April 2006 (Jamae & Johnson, 2009).

Platform architecture

The JBoss application server is based on a microkernel design into which components can be plugged at runtime to extend its behavior (Monson-Haefel & Burke, 2006). At its core is a microkernel-based server that is extremely small in footprint. By utilizing Java Management Extensions (JMX), the microkernel delivers a lightweight

component model that offers hot deployment and advanced class-loading features and full lifecycle management. The JMX allows developers to integrate components like modules, containers, and plug-ins. These components are declared as Management Beans (MBeans) that are loaded into the JBoss. The JMX microkernel manages the Management Beans (MBeans) (Fleury et al, 2005).

The JBoss documentation describes the microkernel as a spinal cord where the JBoss in its minimum configuration is like the brainstem sitting at the top of the spine. The spinal column is the attachment point for all the MBeans that want to talk to the brainstem and to each other over the spinal cord (Richards & Griffith, 2005). Figure 28 shows the architecture of the JBoss microkernel architecture.

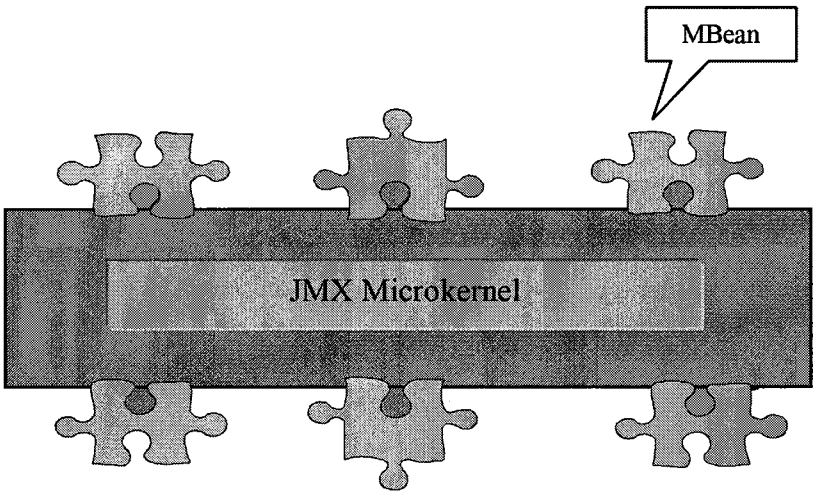

Figure 28. JBoss Application Server microkernel architecture

Since JBoss 4.0.3, the application server started migrating to the microcontainer architecture where it has a small runtime kernel that provides common functionalities for the application server basic components. Services are built by using Plain Old Java Objects (POJO) rather than the MBean. The JMX still plays an important role in JBoss microcontainer architecture because not all services have been ported to the microcontainer such as Java Message Services (JMS) and Java Naming and Directory Interface (JNDI). Therefore, the JMX POJO is one of the primary POJOs defined to and created by the microcontainer. Figure 29 represents the JBoss microcontainer architecture.

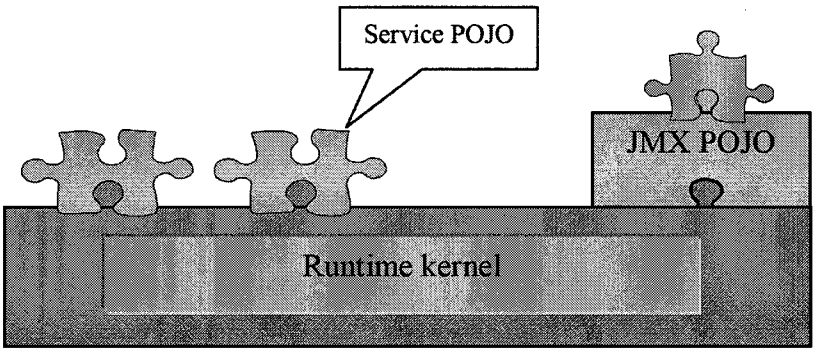

Figure 29. JBoss Application Server microcontainer architecture

Collaboration Mode

All the POJO's and MBeans are considered internal extensions for the JBoss application server. Each POJO and MBean development process is managed by a subproject under the JBoss Application Server project. To ensure that there is a clear

focus and organized approach to the development process, each project has a team consisting of the following[8]:

- Project lead or lead developer, who is responsible for the project overall direction and vision. Project leads are typically employed by Red to ensure the maximum dedication of time and effort to the project and ensure that releases are planned correctly and completed on schedule. The leads has the authority to decide who receives commit access to the source code repository to ensure the quality of the software and the associated documentation remains high and the conceptual integrity of the project is maintained.

- Project Administrators, each project can have one or more administrators, who are users that have normal project permissions and in addition they have the following responsibilities: i) change the full name of the project, ii) change the documentation URL, iii) change the lead developer, iv) change the project description, and v) change, assign, move, close, delete, and set/edit the "due date" and manage the security levels of issues to prevent misuse. The role of project administrator is crucial to the successful management of a project;

---

[8] JBoss governance model details can be found at: JBoss governance model http://www.jboss.org/community/wiki/GovernanceModelContent. Retrieved on Oct 31, 2009.

therefore, the role is typically reserved for the project leads employed by Red Hat.

- Developers, who come from the community and include individuals contributing code and documentation in their spare time as well as employees from Red Hat and other companies who contribute to the project. The developer has the commit access to the source code repository allowing them to make changes on their own. All changes are covered either by an Individual or Corporate Contributor's Agreement that sets out terms and conditions for how they are used.

Table 18 shows the collaboration modes for JBoss POJO and MBeas development. The development process for the JBoss projects is influenced by Red Hat since every project has at least one or two personnel employed by Red Hat to direct the development process and make sure the software produced of high quality. Even the submitted patches have to be reviewed by the project development team before commit to the project source code to ensure the quality[9]. So the governance structure is very hierarchical but the contribution network is open since the rest of the team includes developers that are individuals from the rest of the community.

---

[9] Details for the JBoss development process can be found at: http://docs.jboss.org/process-guide/en/html/productizing.html#d0e345. Retrieved on Oct 31, 2009.

Table 18. Collaboration modes for JBoss Application Server

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open	JBoss POJO's or MBeans		
	Closed	JBoss runtime kernel		

Community structure

Figure 30 represent the structure of the developer community for JBoss. The inner circle represents the Red Hat developers that are responsible for developing the JBoss application server runtime kernel. The outer circle represents the POJO and MBeans developer community. Since each project has a Red Hat employee as a project lead, then the community contains a mix of external developers as well as Red Hat project leads. The "dots" in the outer circle represent the project leads.

None of the previous cases had that level of platform owner intervention in the development process. Although in the OpenOffice.org case had a Sun Microsystems representative among the OpenOffice.org Community Council but not at the level of the projects.

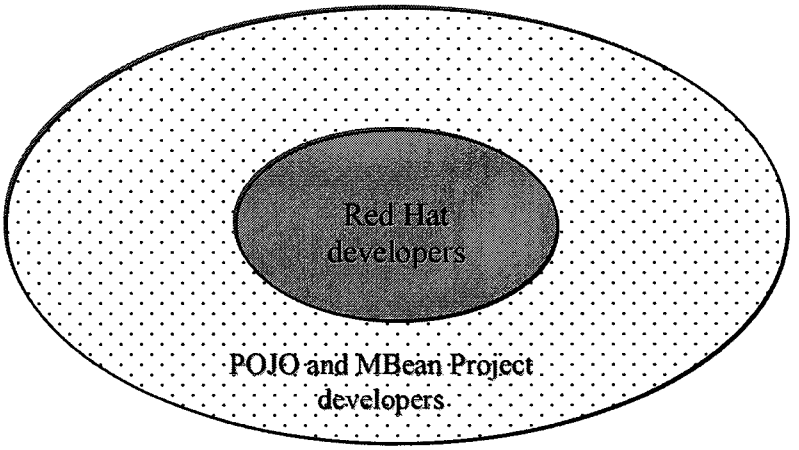

Figure 30. JBoss community structure

Regulatory instruments

Red Hat controls the quality of the JBoss developed components core by limiting the development process to its own team of developers. For other projects, Red Hat controls the quality by having a project lead works full time to manage the project and supervise the development process.

There are number of toolkits that help developers to test JBoss developed components such as JBoss Testsuite tool (http://www.jboss.org/community/wiki/TestsuiteGuide) and Hudson tool (http://hudson.jboss.org/hudson/). The Testsuite is a collection of JUnit tests which require a running JBoss instance for in-container testing and the Hudson is a tool for monitoring continuous builds of all JBoss.org projects.

4.11 JBoss Application Server, commercial version

Red Hat provides a commercial version of the JBoss Application Server. The commercial JBoss application server is a certified and tested copy of the open source application server bundled within the JBoss Enterprise Application Platform. Red Hat offers two types of support for the JBoss commercial version: i) it supports the features included in the commercial release, and ii) offers support to developed feature by external parties who are part of its various developer partnership programs such as the Independent Software Vendor (ISV) developer program.

Platform architecture

The architecture for the JBoss commercial release is based on the JBoss open source application server and Red Hat incorporate its on components and updates for the commercial version.

Collaboration Mode

The development process for the commercial version is limited to Red Hat developers. They carry out the tasks of bug fixing, testing, upgrading components, and support for the certified version. Table 19 shows the collaboration mode for the commercial version of the JBoss application server. Despite the tight control that Red Hat imposes on the development process, the commercial version inherits the flow of ideas from the JBoss open source project.

Table 19. Collaboration modes for JBoss commercial version

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open			
	Closed	JBoss commercial version components (core, POJO's and MBeans)		

Community structure

The developer community for the JBoss commercial version consists of the Red Hat developers for the runtime kernel, POJO's and MBeans containers (or the internal extensions). Figure 31 shows the community structure for the commercial release as it includes the Red Hat developers only.

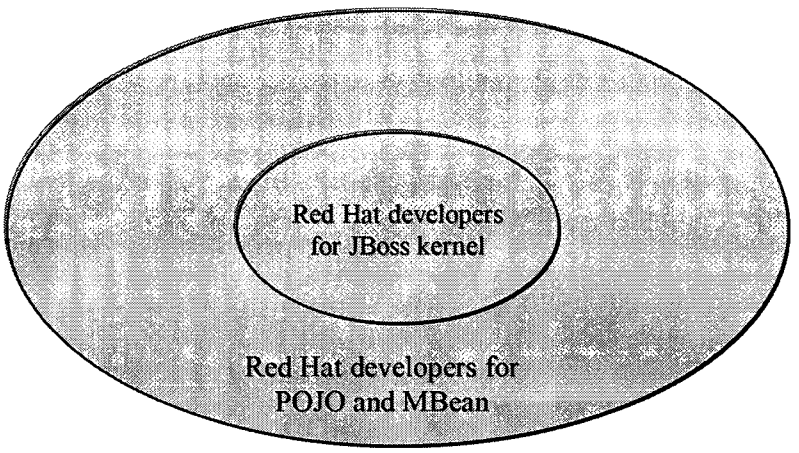

Figure 31. Community structure for the JBoss commercial version

Regulatory instruments

There is not a publicly available toolkit for developing and testing the commercial release of the JBoss server because the development process is limited to Red Hat employees. So Red Hat do not invest in publicly toolkits for developing the JBoss server extensions but it provides toolkits and development support programs for JBoss technology based applications.

4.12 MySQL Relational Database Server

MySQL is the most popular open source relational database management system, is an attractive alternative to higher-cost relational systems from commercial vendors. MySQL is currently owned and developed by Sun Microsystems. MySQL relational database server was born out of an internal company project by employees of the

Sweden-based TcX DataKonsult AB. MySQL server was first released to the public at the end of 1996 (Kruckenberg & Pipes, 2005; Shibuya & Tamai, 2009).

Platform architecture

MySQL server architecture does not represent the traditional modular architecture, it consists of a web of interrelated function sets, which work together to fulfill the various needs of the database server. Each related set of functionality known by the term "subsystem", rather than component. The subsystems are quite independent of each other and they accept information from and feed data into the other subsystems of the server through a well-defined function application programming interface (API). The overall organization of the MySQL server architecture is a layered, but not particularly hierarchical, structure. It consists of a base function library and a select group of subsystems that handle the lower-level responsibilities. These libraries and subsystems serve to support the abstraction of the storage engine systems, which feed data to requesting client programs. The client programs communicate with the engine abstraction through a well defined Client Application programming interface (API) (Kruckenberg & Pipes, 2005). Figure 32 represents a general description for the MySQL subsystem architecture. Despite the fact of its semi-modular architecture, MySQL server is the most popular database servers.

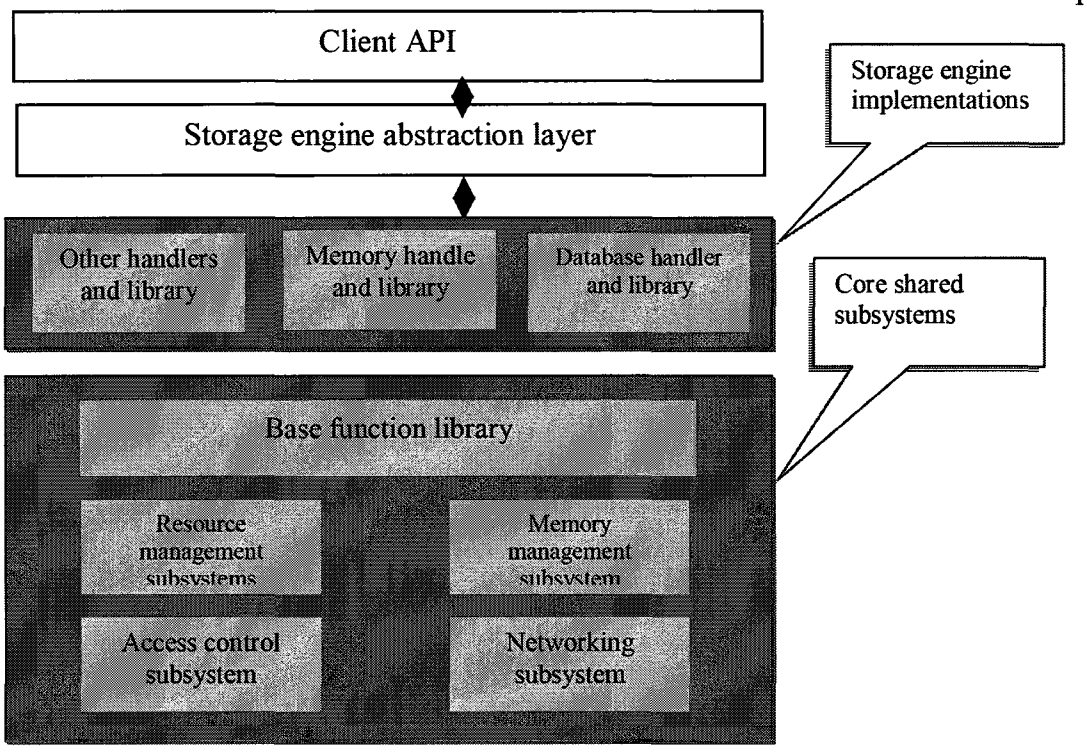

Figure 32. MySQL server subsystem architecture

In January 2009, the refactoring project was created with a goal to improve MySQL codebase modularity, reduce the number of bugs introduced with new features, and provide a pluggable architecture that facilitates third-parties contribution.

Collaboration Mode

MySQL is owned by Sun Microsystems, and the development process for the server is highly controlled by the platform owner. External developers are allowed to contribute code patches and propose new feature but they are not permitted to become part of the

MySQL developer team (Shibuya & Tamai, 2009). The governance structure for MySQL development is hierarchical, because of Sun influence, with a closed network since the process is restricted to MySQL core developers. Table 20 represents the collaboration mode for the MySQL server.

Table 20. Collaboration modes for MySQL extensions

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open			
	Closed	MySQL RDBS subsystems		

Community structure

The developer community of MySQL consists of the MySQL core team that is responsible for developing and managing MySQL codebase. External contribution take different forms such as submitting code patches, suggesting features, testing and benchmarking, and other tasks. In Figure 33 represents the MySQL community structure, where the inner circle represents MySQL developer team and the outer circle represents the external contributor to MySQL server project.

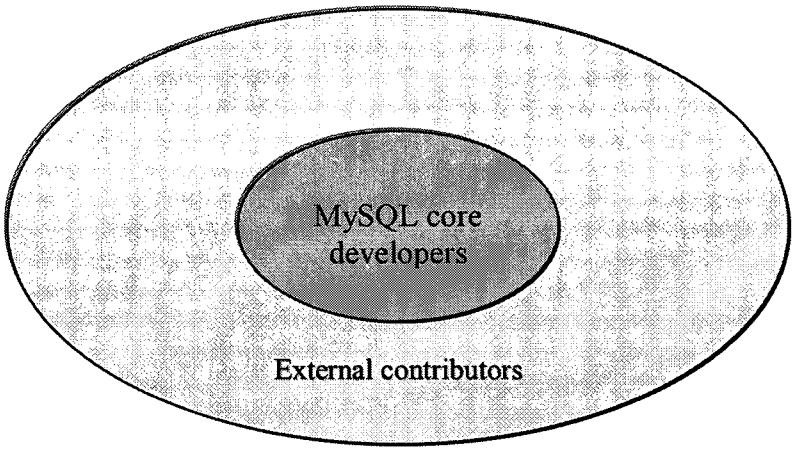

Figure 33. MySQL community structure

Regulatory instruments

For quality assurance, MySQL depends on its vibrant community for testing and reporting bugs (Watson et al., 2008). MySQL provides the community with tools for testing such as Test Maker for system testing and VisiBones's MySQL Unit Tests for unit testing.

4.13 Summary of the platform cases

Tables 21 and 22 represent the summary for the information that was drawn from the 12 platform cases in the sample. Table 21 includes the following information: platform name, platform architecture, extension form (i.e. plug-in, module, function subsystem, etc), extension type (i.e. internal, external), governance structure (hierarchical - H, hierarchical / flat – H/F, flat - F), network openness (open, closed), extension quality (high, medium, low), and flow of idea (high, medium, low). Table 22 represent includes the following information: platform name, regulatory instrument (pricing, toolkit, sandbox, extension development process, legal instrument), extension quality (high, medium, low), and flow of idea (high, medium, low).

In Table 21, since all platforms have a core component with hierarchical governance structure and closed network; therefore we only included the extension type in the table.

Table 21. Summary of platform cases and related architecture, extension type, governance structure, network, extension quality, and flow of ideas

No	Platform	Platform architecture and extensions form	Extension type	Governance structure	Network openness	Extension quality	Flow of ideas
1	Eclipse platform	OSGi based, plug-in	Internal	H/F	Open	High	Medium to high
			External	F	Open		
			Commercial	H	Closed		
2	IBM WebSphere Application Server	OSGi based, plug-in	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
3	Firefox	Modular based on Mozilla platform, extension module	Internal	H/F	Open	High	Medium to high
			External	F	Open		
			Commercial	H	Closed		
4	Spring framework, open source	OSGi based, module	Internal	H	Open	High	Low to medium
			Commercial	H	Closed		
5	Spring framework, commercial	OSGi based, module	Internal	H	Open	Medium to high	Low
			Commercial	H	Closed		
6	Apache HTTP Server	Microkernel based, module	Internal	H/F	Open	High	Medium to high
			External	F	Open		
			Commercial	H	Closed		
7	IBM HTTP Server	Microkernel based, module	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
8	Openoffice. org	Component based, Uno	Internal	H/F	Open	Medium to high	Medium to high
			External	F	Open		
			Commercial	H	Closed		
9	StarOffice	Component based, Uno	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
10	JBoss Application Server	Microcontainer based, microcontainer and MBeans	Internal	H	Open	High	Medium
11	JBoss Application Server, Enterprise edition	Microcontainer based, microcontainer and MBeans	Internal	H	Closed	Medium to high	Low
12	MySQL DB Server	To be refactored to a plug-in based, plug-in	Internal	H	Closed	Medium to high	Low to medium

Table 22. Summary of platform cases and related regulatory instruments, extension quality, and flow of ideas

No	Platform	Regulatory instruments				Extension quality	Flow of ideas
		Pricing	Toolkits	Sandbox	Extension development process		
1	Eclipse platform	Yes	Yes	No	Incubation	High	Medium to high
2	IBM WebSphere Application Server	Yes	Yes	No	None	Medium to high	Medium
3	Firefox	No	Yes	Sandbox review	None	High	Medium to high
4	Spring framework, open source	No	Yes	No	Extensions project	High	Low to medium
5	Spring framework, commercial	Yes	Yes	No	None	Medium to high	Low
6	Apache HTTP Server	Yes	Yes	Yes	Incubation	High	Medium to high
7	IBM HTTP Server	No	Yes	No	None	Medium to high	Medium
8	Openoffice.org	No	Yes	No	Extensions project and quality assurance procedure	Medium to high	Medium to high
9	StarOffice	No	Yes	No	None	Medium to high	Medium
10	JBoss Application Server	No	Yes	No	Incubation projects	€€€High	Medium
11	JBoss Application Server, Enterprise edition	Yes	Yes	No	None	Medium to high	Low
12	MySQL DB Server	No	Yes	No	None	Medium to high	Low to medium

5 RESULTS

This chapter is organized in two sections. Section 5.1 represents the different governance models followed by platform owners to control the quality of extensions. Section 5.2 represents the regulatory instruments used by platform owners to protect the platform integrity.

5.1 Platform extension governance models

Based on the data that was drawn from the twelve platform cases in the sample, three governance models were observed that were used by platform owners in the case of developing platform extensions. The differentiation between each model and another is based on the following factors: platform architecture (i.e. extension type, modularity), collaboration mode, and community structure. In the next three sections we are going to discuss the properties of each model in details.

5.1.1 Model A

Table 23 represents the platform cases that adopt the governance model A. In Table 23 we can see that five cases had the same implementations to control the platform extension development process. Model A facilitates the platform owner control over the development process and the quality of the product. Also it directs efforts towards achieving the platform owner desire in developing certain requirements. But the tight control of the development process limits the number of the ideas in the platform

network. By examining the data in Table 23, we can see that model A has the following characteristics:

Table 23. Cases represent governance model A

No	Platform	Community structure	Extension type*	Governance structure	Network openness	Extension quality	Flow of ideas
1	Spring framework, open source	Core, internal	Internal	H	Open	High	Low to medium
			commercial	H	Closed		
2	Spring framework, commercial	Core, internal	Internal	H	Open	Medium to high	Low
			Commercial	H	Closed		
3	JBoss Application Server	Core, internal	Internal	H	Open	High	Medium
4	JBoss Application Server, Enterprise edition	Core, internal	Internal	H	Closed	Medium to high	Low
5	MySQL DB Server	Core, internal	Internal	H	Closed	Medium to high	Low to medium

* All platforms have a core component with hierarchical governance structure and closed network; therefore we only included the extension type in the table

Platform architecture

From Table 23, we can see that the platform has a core and internal extensions only (as shown in figure 34). The internal extensions are essential to the platform functionality and their quality highly affects the platform integrity. In SpringSource case the commercial extensions are the same as the internal with a difference in the collaboration mode.

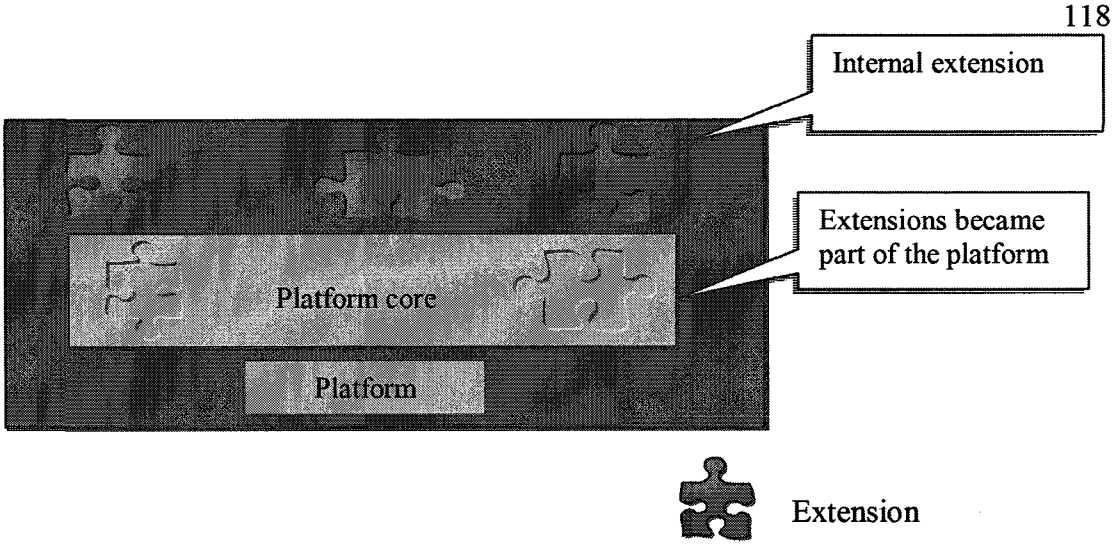

Figure 34. Platform architecture for model A

Collaboration mode

In Table 23, all the five cases have a hierarchical governance structure and the network of developers is closed for internal extensions. Table 24 represents the collaboration mode for model A. Because of the platform sensitivity to the quality of the internal extensions, platform owners highly influence and control the process of developing internal extensions.

Table 24. Collaboration mode for model A

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open	Internal extensions		
	Close	Core components and commercial extensions		

Community structure

In Table 23, the developer community has the core team developers, which is normally hired by the platform owner, is responsible for: i) the platform core technology development, ii) maintaining the platform codebase, and iii) supervising and monitoring the development process for the internal extensions. The rest of the community is the individual developers that work on developing the internal extensions. Internal developers has to earn their position among the platform developer community by submitting high quality code and following the guidelines set by the platform owner or external parties afford membership fees to join the platform development club. Figure 35 represent the community structure for model A.

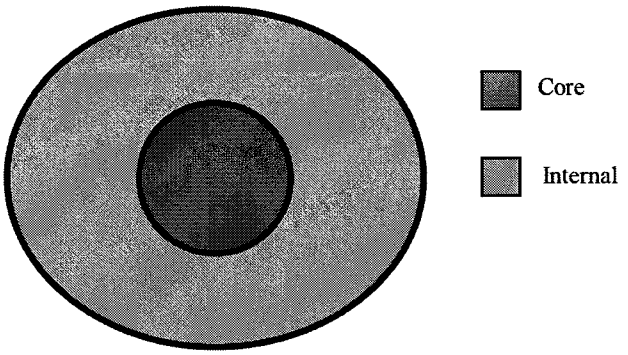

Figure 35. Community structure for model A

5.1.2 Model B

Table 25 represents the platform cases in the sample that adopt the governance model B. In Table 25 we can see that three cases had the same implementations to control the platform extension development process. Model B allows unlimited number of ideas and with high flow of ideas the unpredictability could lead to unusual use of the platform functionalities that can triggers the innovation in the community. The downside of model B is the unpredictability and the difficulty to control the community of developers; therefore, it is hard to guarantee the quality of the product (i.e. platform extensions). By examining the data in Table 25, we can see that model B has the following characteristics:

Table 25. Cases represent governance model B

No.	Platform	Community structure	Extension type*	Governance structure	Network openness	Extension quality	Flow of ideas
1	IBM WebSphere Application Server	Core, external	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
2	IBM HTTP Server	Core, external	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
3	StarOffice	Core, external	Internal	H	Closed	Medium to high	Medium
			External	F	Open		

* All platforms have a core component with hierarchical governance structure and closed network; therefore we only included the extension type in the table included the extension type in the table

Platform architecture

From Table 25, we notice that the platform has a platform core, internal, and external extensions only (as shown in figure 36). The internal extensions are essential to the platform functionality as they become part of the platform core and their quality highly affects the platform integrity. On the other hand, external extensions are not essential to the platform functionality but their quality affects the platform integrity and the other extensions.

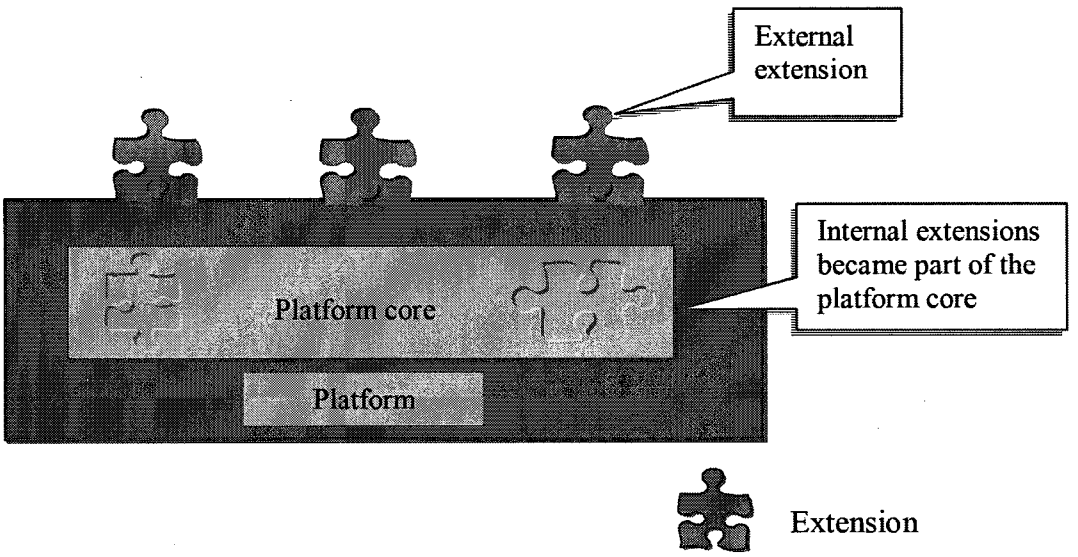

Figure 36. Platform architecture for model B

Collaboration mode

In Table 25, all the three cases have a hierarchical governance structure and the network of developers is closed for internal extensions. Because of the platform sensitivity to the quality of the internal extensions, platform owners highly influence and control the process of developing internal extensions. In the case of the external extension, the level of ideas is important more that the extension quality itself. So we notice in Table 25 that external extensions have a flat governance structure with an open network of developers. Table 26 represents the collaboration mode for model B.

Table 26. Collaboration modes for Mode B

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open			External extensions
	Close	Core components, internal extensions		

Community structure

In Table 25, the developer community has the core team developers like model A. The core team normally hired by the platform owner and is responsible for: i) the platform core technology development and its internal extensions, ii) maintaining the platform codebase, and iii) providing support for the core components and the internal extensions. The rest of the community is the external developers, which they are individuals and external parties, work on the development external extensions. Figure 37 represent the community structure for model B.

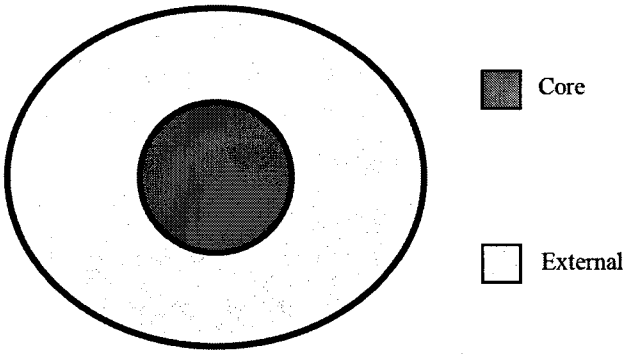

Figure 37. Community structure for model B

5.1.3 Model C

Table 27 represents the cases that adopt the governance model C. In Table 27 we can see that four cases had the same implementations to control the platform extension development process. In model C the Platform owners have control over who develop what in the community and control the quality of developed product on different levels. Platform owners play the orchestrator role for the flow of ideas in the community. It requires plenty of resources (e.g. time, staff, technical ablates, and special expertise) to perform the tasks of managing the community in model C. By examining the data in Table 27, we can see that model B has the following characteristics:

Table 27. Cases represent governance model C

No.	Platform	Community structure	Extension type*	Governance structure	Network openness	Extension quality	Flow of ideas
1	Eclipse platform	Core, internal, external	Internal	H/F	Open	High	High
			External	F	Open		
			Commercial	H	Closed		
2	Firefox	Core, internal, external	Internal	H/F	Open	High	High
			External	F	Open		
			Commercial	H	Closed		
3	Apache HTTP Server	Core, internal, external	Internal	H/F	Open	High	High
			External	F	Open		
			Commercial	H	Closed		
4	Openoffice.org	Core, internal, external	Internal	H/F	Open	Medium to high	High
			External	F	Open		
			Commercial	H	Closed		

* All platforms have a core component with hierarchical governance structure and closed network; therefore we only included the extension type in the table

Platform architecture

From Table 27, we notice that the platform has a platform core, internal, and external extensions (as shown in figure 38). The internal extensions are essential to the platform functionality as they become part of the platform core and their quality highly affects the platform integrity. On the other hand, external extensions are not essential to the platform functionality but their quality affects the platform integrity and the other extensions.

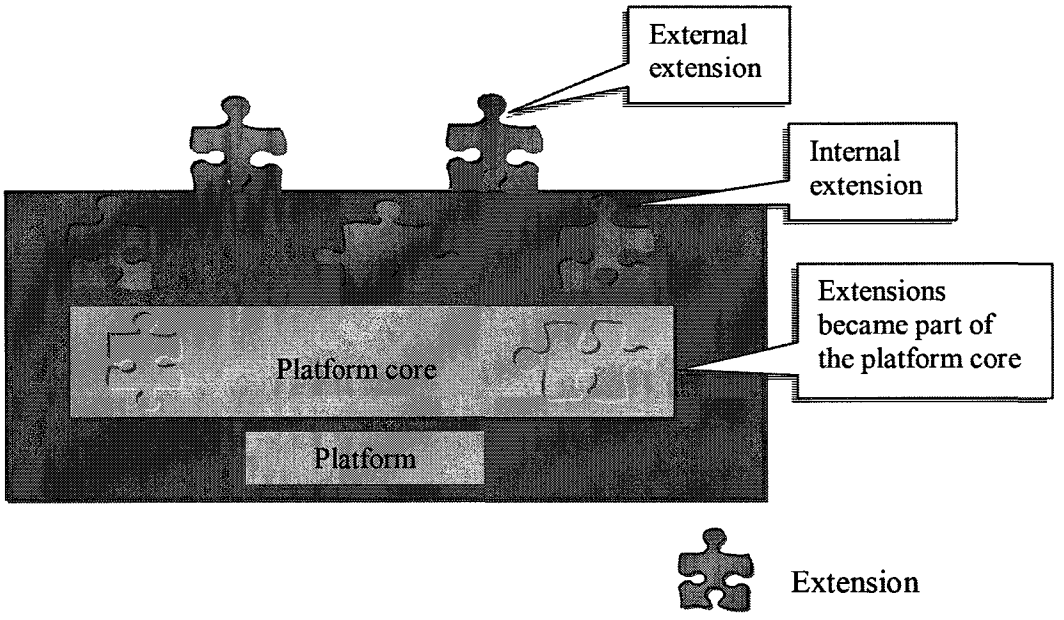

Figure 38. Platform architecture for model C

Collaboration mode

In Table 27, all the four cases have a hierarchical governance structure and the network of developers is closed for the core and the commercial extensions. In Table 27 we notice that the internal extensions in model C have a hierarchical/flat governance structure and open network of developers. The external extensions have a flat governance structure with an open network of developers. Table 28 represents the collaboration mode for model C.

Table 28. Collaboration modes for model C

		Governance structure		
		Hierarchical	Hierarchical/Flat	Flat
Network openness	Open		Internal extensions	External extensions
	Close	Core components and commercial extensions		

Community structure

In Table 27, the developer community has the core team developers like model A. The core team normally hired by the platform owner and is responsible for: i) the platform core technology development and its internal extensions, ii) maintaining the platform codebase, and iii) providing support for the core components and the internal extensions. The rest of the community consists of the internal developers who work on the internal extensions and the external developers, which they are individuals and external parties, work on the development external extensions. Figure 39 represent the community structure for model C.

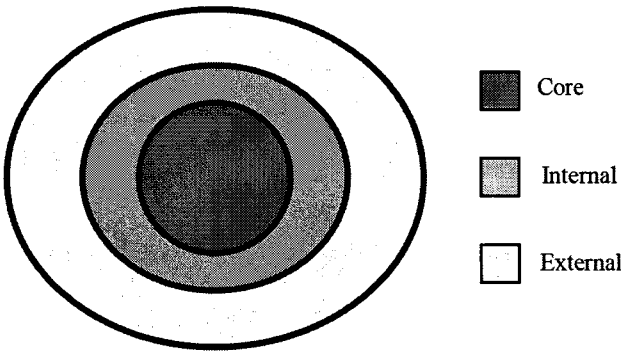

Figure 39. Community structure for model C

5.1.4 Summary of governance models

Table 29 presents the summary for the platforms cases and the associated governance models, extension types, governance structure, network openness, extensions quality, and flow if ideas. Table 30 represents the summary of the governance model and the associated extension types, governance structure, network openness, extensions quality, and flow if ideas. Finally Table 31 represents a summary for the advantage and disadvantages for the governance models.

Table 29. Summary of the platforms and governance models

No.	Platform	Governance model	Extension type	Governance structure	Network openness	Extension quality	Flow of ideas
1	Eclipse platform	Model C	Internal	H/F	Open	High	Medium to high
			External	F	Open		
			Commercial	H	Closed		
2	IBM WebSphere Application Server	Model B	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
3	Firefox	Model C	Internal	H/F	Open	High	Medium to high
			External	F	Open		
			Commercial	H	Closed		
4	Spring framework, open source	Model A	Internal	H	Open	High	Low to medium
			commercial	H	Closed		
5	Spring framework, commercial	Model A	Internal	H	Open	Medium to high	Low
			Commercial	H	Closed		
6	Apache HTTP Server	Model C	Internal	H/F	Open	High	Medium to high
			External	F	Open		
			Commercial	H	Closed		
7	IBM HTTP Server	Model B	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
8	OpenOffice. Org	Model C	Internal	H/F	Open	Medium to high	Medium to high
			External	F	Open		
			Commercial	H	Closed		
9	StarOffice	Model B	Internal	H	Closed	Medium to high	Medium
			External	F	Open		
10	JBoss Application Server	Model A	Internal	H	Open	High	Medium
11	JBoss Application Server, Enterprise edition	Model A	Internal	H	Closed	Medium to high	Low
12	MySQL DB Server	Model A	Internal	H	Closed	Medium to high	Low to medium

Table 30. Summary of governance models

Collaboration Model	Extension types	Governance structure	Network openness	Extension quality	Flow of ideas
Model A	Internal	Hierarchical	Closed/open	Medium to high	Low to medium
	Commercial	Hierarchical	Closed		
Model B	Internal	Hierarchical	Closed	Medium to high	Medium
	External	Flat	Open		
Model C	Internal	Hierarchical / flat	Open	High	Medium to high
	External	Flat	Open		
	Commercial (internal or external)	Hierarchical	Closed		

Table 31. Governance models advantages and disadvantages

Collaboration Model	Advantage	Disadvantage
Model A	• Easy to control the development process. • Easy to control the quality of the product. • Direct efforts towards achieving the platform owner desire in developing certain requirements.	• Limited number of ideas.
Model B	• Large number of ideas. • Unpredictability could lead to unusual use of the platform functionalities that can trigger innovation in the community.	• Unpredictable community of developers. • Difficult to control the community of developers. • There is no way to guarantee the quality of the product.

Collaboration Model	Advantage	Disadvantage
Model C	• Platform owners have control over who develops what in the community. • Platform owners can control the quality of developed product on different levels Platform owner's is the orchestrator for the flow of ideas in the community.	• Requires effort from the organization to oversee the different parties involved.

5.2 Regulatory instruments

In order to support and control the development process of the platform extensions, the platform owners use a number of regulatory instruments in conjunction with the governance model. These regulatory instruments enable the platform owners to manage information transactions in the platform network, support the process of developing extensions, and control the quality of the product by providing the community with tools to develop, test, integrate extensions, and share the experience among the platform members. These instruments help platform owners to establish barriers of entry to the developer community such as pricing or the development process. Instruments like platform architecture and toolkits help to create technical boundaries between the platform and the developer community. Figure 40 show the position of the regulatory instruments as a boundary between two levels of developers in the community.

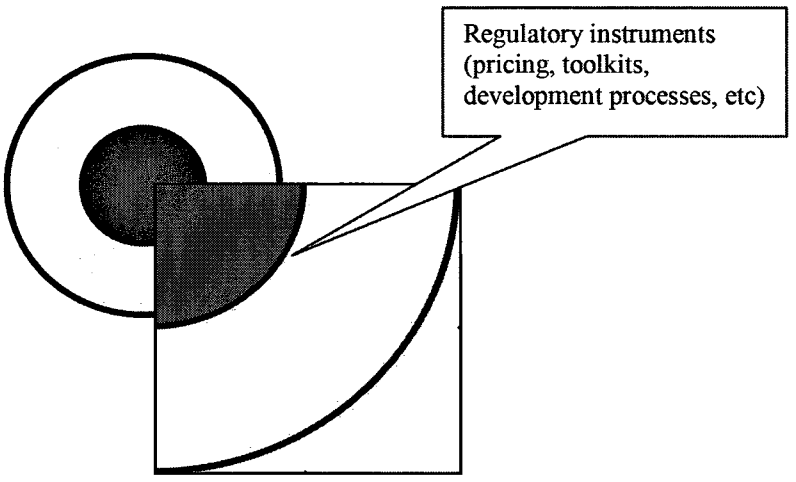

Figure 40. Regulatory instruments as a boundary between two levels in the platform developer community

Table 32 represents a summary for the platform cases and the associated regulatory instruments used, governance model, extension quality, and flow of ideas; and table 33 represents a summary for the regulatory instruments types, number of platform case adopted the instrument, associated governance model. Our focus is the instruments that are affecting the quality of the extension, therefore, the platform architecture is not included as an instruments since all the platforms subject to the study are extensible and we did not include the none extensible cases and all types of extensions (i.e. internal and external) are technically equal as they use the same interfaces and protocols to communicate with the platform and other extensions.

In Table 32, four types of regulatory instruments were used to regulate the information transactions and control the quality of the extensions (i.e. pricing, toolkits, sandboxes, and extension development process.

Table 32.Summary of platform cases and the associated regulatory instruments and governance model

No	Platform	Regulatory instruments				Governance model	Extension quality	Flow of ideas
		Pricing	Toolkits	Sandbox	Extension development process			
1	Eclipse platform	Yes	Yes	No	Incubation process	Model C	High	Medium to high
2	IBM WebSphere Application Server	Yes	No	No	None	Model B	Medium to high	Medium
3	Firefox	No	Yes	Yes	None	Model C	High	Medium to high
4	Spring framework, open source	No	No	No	Extensions project	Model A	High	Low to medium
5	Spring framework, commercial	Yes	No	No	None	Model A	Medium to high	Low
6	Apache HTTP Server	Yes	No	No	Incubation process	Model C	High	Medium to high
7	IBM HTTP Server	No	Yes	No	None	Model B	Medium to high	Medium
8	Openoffice.org	No	Yes	No	Extensions project and quality assurance procedure	Model C	Medium to high	Medium to high
9	StarOffice	No	Yes	No	None	Model B	Medium to high	Medium
10	JBoss Application Server	No	Yes	No	None	Model A	High	Medium
11	JBoss Application Server, Enterprise edition	Yes	No	No	None	Model A	Medium to high	Low
12	MySQL DB Server	No	Yes	No	None	Model A	Medium to high	Low to medium

Data from Table 32 was used to construct Table 33 to present the commonality of an instrument certain type and the relation between the instrument types and the governance model used. In table 33 we can see that: i) toolkit instrument is the most common instrument for all governance models as it was adopted by 7 out 12 cases, ii) the sandbox instrument is still at an early stage where only 1 case out of the 12 is using it, iii) the development process is mainly associated with governance model C, and iv) pricing is a common instrument for all governance models as it was adopted by 5 cases out of 12.

Table 33. Summary of regulatory instruments types, number of platform cases, and associated governance model

No.	Regulatory instrument	Number of platforms	Governance model (s)
1	Toolkit	7	Model A -2 Model B-2 Model C-3
2	Pricing	5	Model A- 2 Model B-1 Model C-2
4	Development process	4	Model A -1 Model B-0 Model C-3
3	Sandbox	1	Model A -0 Model B-0 Model C-1

6 DISCUSSION

6.1 Strategies for managing the quality of platform extensions

There is no single approach that platform owners can follow to manage the quality of extensions. Each approach is associated with non-trivial trade-offs in terms of governance, openness, quality and flow of ideas. Furthermore, different levels of governance and openness may be applied to different types of extensions. Internal extensions, which are more widely used and usually deployed together with the platform core, are often more tightly controlled than external extensions, which are developed to meet more specialized needs. In the case of internal extensions, there is a significant impact of low quality of those extensions on the platform and on each other; a reduced flow of new ideas is traded off against higher quality. As for the external extensions, it is more important to allow new ideas to develop than to monitor their quality. Yet, the distinction between internal and external extensions is not fixed; over its life an extension may change its type.

For the extensions quality management, we refine the Pisano & Verganti (2008) framework. Our framework has four independent dimensions: governance structure, network openness, modularity of platform architecture, and regulatory instruments and two dependent dimensions: extensions quality and flow of ideas.

1. A modular platform architecture enables and increases involvement of external parties in the platform development process. It also enables platform owners to

keep control of the platform architecture as a powerful barrier against competing architectures and protects the integrity of the platform itself (Cusumano & Gawer, 2002; Messerschmitt & Szyperski, 2003). Also having a good modular design reduces the range and scope of interaction between components, which reduces the amount and range of cycling that occurs in the development process, that results in fewer cycles and less time spent on the development process. As a result modularity will increase the probability of success and the quality of the final product.

2. The governance structure of a network can be either hierarchical, flat, or a hybrid between these extremes. In a hierarchical governance structure, the platform owner both defines problems and selects which solutions are adopted, whereas in a flat structure both problems and solutions are decided on by a community. The case between those two extremes is a hybrid of hierarchical and flat structures: while problems are decided on by the community, solutions are selected by the platform owner. The framework by Pisano & Verganti (2008) did not include such a category, leading them to classify both Linux and Apple's iPhone as examples of innovation communities, although they are very different in nature.

3. Openness of a platform network refers to the degree to which participation in the network is open. In an open network, any party (partners, customers, or even competitors) can contribute to the platform. Open source projects are examples of this type of network. In a closed network, the platform owner selects who can

participate based on the capabilities and resources required for the innovation (Pisano & Verganti, 2008).

4. Regulatory instruments are used by platform owners to regulate information transactions among members in their network as well as to control the quality of components developed by third parties (Boudreau & Hagiu, 2009; von Hippel; Bosch & Bosch-Sijtsema, 2009). Regulatory instruments include pricing and non-pricing instruments: pricing, toolkits, sandboxes, and development process.

5. The flow of ideas determines on the extent and richness of the information exchanged through the network. Restricted flow of ideas is directed and obscured with exception to the information needed to complete a specific task. It is effective, when tasks are well-defined and it is easy to determine who is most suited to perform them. However, when a company does not know who to ask for a solution to a problem, or which of several solution is the most likely to succeed, it is better not to restrict information flow and open up for external sources of information.

6. Quality refers to the quality of extensions and the platform. Composability is a key quality attribute that is important to ensure the platform integrity. Quality assurance of individual extension is similar to unit testing, and is focused on the extension in isolation. Ensuring integrity is similar to integration testing: while all extensions may demonstrate the expected functionality in isolation, they may not

work with one another, or may even break functionality in the platform core (Koufteros, et al., 2005).

Based on the discussions above we propose the framework in Figure 41 which illustrates how a platform owner makes key structural decisions of the governance model for the community of developers of platform extensions. Decisions regarding platform modularity, governance structure, network openness, and regulatory instruments (Thomke & von Hippel, 2002; Gawer & Cusumano, 2008; Pisano & Verganti, 2008; West & O'Mahony, 2008; Boudreau & Hagiu, 2009; Hagiu, 2009) affects the level of community contribution, measured by the quality the platform extensions and the levels of innovation, measured by the flow of ideas in the platform network. The literature stresses the importance of innovation and the quality of products where high innovation levels imply high flow of ideas that accelerates the development process and improve the quality capabilities of the products (Chesbrough, 2003; Herzog, 2008). Table 34 presents the list of the propositions for the platform extensions governance model in relation with extensions quality and flow of ideas.

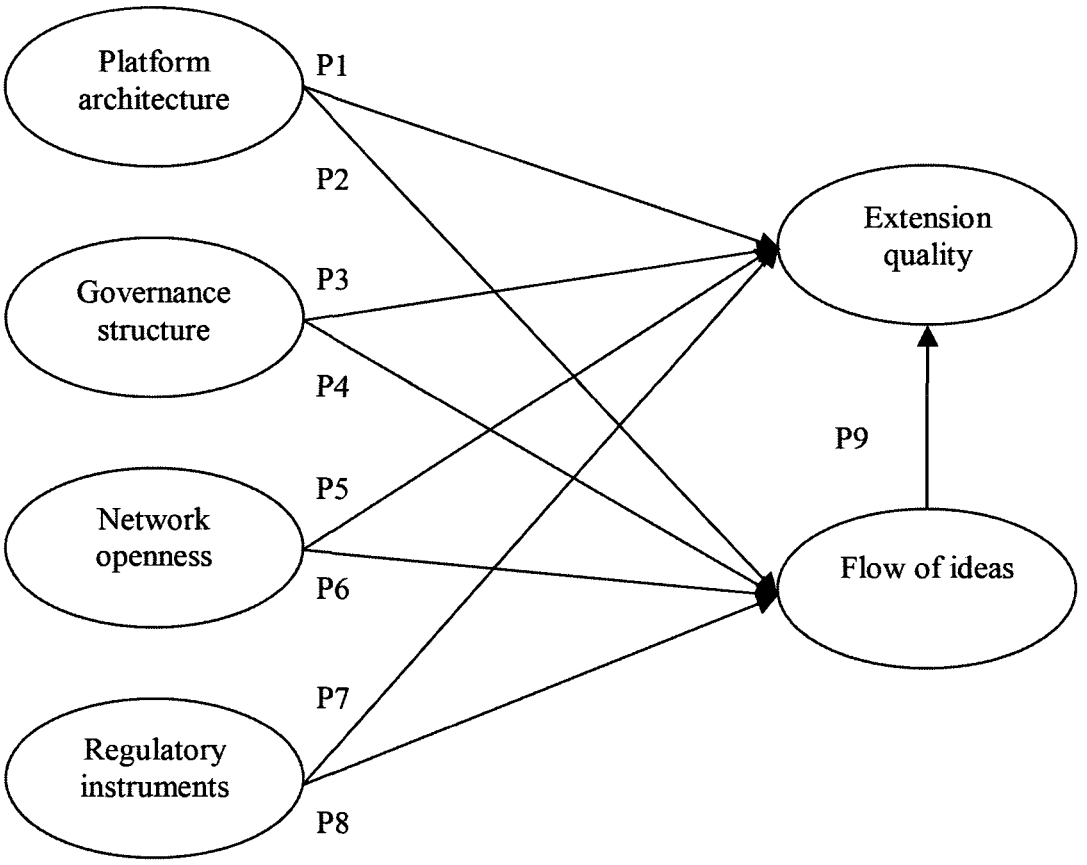

Figure 41. Framework for platform extension development governance

Table 34. List of governance model propositions

P1: *Platform architecture degree of modularity is positively associated with the extension quality. This implies high degree of modularity is associated with medium to high extension quality.*

P2: *Platform architecture degree of modularity is positively associated with the flow of ideas. This implies high degree of modularity is associated with medium to high flow of ideas.*

P3: *Flatness of governance structure is negatively associated with the extension quality. This implies:*

- *Flat governance structure is associated with low extension quality.*
- *Hierarchical governance structure is associated with medium to high extension quality.*
- *Hierarchical/flat governance structure is associated with high extension quality.*

P4: *Flatness of governance structure is positively associated with the flow of ideas. This implies:*

- *Flat governance structure is associated with high flow of ideas.*
- *Hierarchical governance structure is associated with low flow of ideas.*
- *Hierarchical/flat governance structure is associated with high to medium flow of ideas.*

P5: *Network openness is positively associated with the extension quality. This implies open network is associated with high extension quality.*

P6: *Network openness is positively associated with the flow of ideas. This implies open network is associates with high flow of ideas.*

P7: *Regulatory instruments may be positively or negatively associated with the extension quality based on the type of instrument.*

P8: *Regulatory instruments may be positively or negatively associated with the flow of ideas based on the type of instrument.*

P9: *High level of innovation is positively associated with the extension quality.*

6.2 Regulatory instruments for managing the quality of platform extensions

As the platform owners have different varieties of instruments to manage the quality of platform extensions. Form the results in section 5.2, the regulatory menu of instruments include: pricing, toolkits, sandboxes, and development process (incubation process). In the next four sections we are going to discuss these instruments and their role in control the quality of the extensions and tier effect on the flow of ideas in the network.

Pricing or membership refers the pricing schema set by the platform owner to charge third-party developers for gaining access rights to the platform, information, and service. Pricing and membership programs enable platform owners to filter inflow of the ideas as well as the quality in the network. Although using the pricing instrument enables the platform owner to control the flow of ideas but there is no guarantee to eliminate bad quality contribution as it depends on the pricing structure and the platform owner's need to access external resource (Hagiu, 2008).

We introduced the notion of toolkits and sandboxes in section 2.2.2. A toolkit is form of software infrastructure that reduces the time and effort required to develop, provision, and operate extensions as well as they contribute to quality. To ensure the quality of extensions, the Mozilla project offers a toolkit consisting of several tools to test the performance and quality of Firefox extensions. The Eclipse platform also

provides a plug-in development environment (PDE), a comprehensive series of build and user interface creation tools, and tools for API maintenance.

A sandbox is a type of a toolkit that allows extensions to be tested in their actual deployment environment. For example, Firefox provides a sandbox review process on its Firefox add-on site, where extension developers can test their extension before it is reviewed for general use.

The incubation process was another method utilized by platform owners to control the quality of extensions developed by external parties. The incubation process enables the platform owners to filter the flow of idea in the internal extensions community of contributors (Duenas et al., 2007). For some platforms such as Mozilla, the incubator is a working directory that is considered a testing ground for experimenting new ideas and a workspace where lead developers or module owners work with inexperienced developers.

Based on the discussions above we propose the framework in Figure 42 to show the relationship between the regulatory instruments and the quality of extensions and the flow of ideas. Figure 42 shows how different choices of the regulatory instruments such as pricing, toolkits, sandboxes, and extension development process (Thomke & von Hippel, 2002; Duenas et al., 2007; Gawer & Cusumano, 2008; Hagiu, 2008, Boudreau & Hagiu, 2009; Hagiu, 2009) affects the level of community contribution, measured by the quality of the platform extensions and level of innovation, measured

by the flow of ideas in the platform network. And finally the influence of innovation level on the quality of products where high innovation levels imply high flow of ideas that accelerates the development process and improve the quality capabilities of the products (Chesbrough, 2003; Herzog, 2008). Table 35 presents the list of the propositions for the regulatory instruments in relation with extensions quality and flow of ideas.

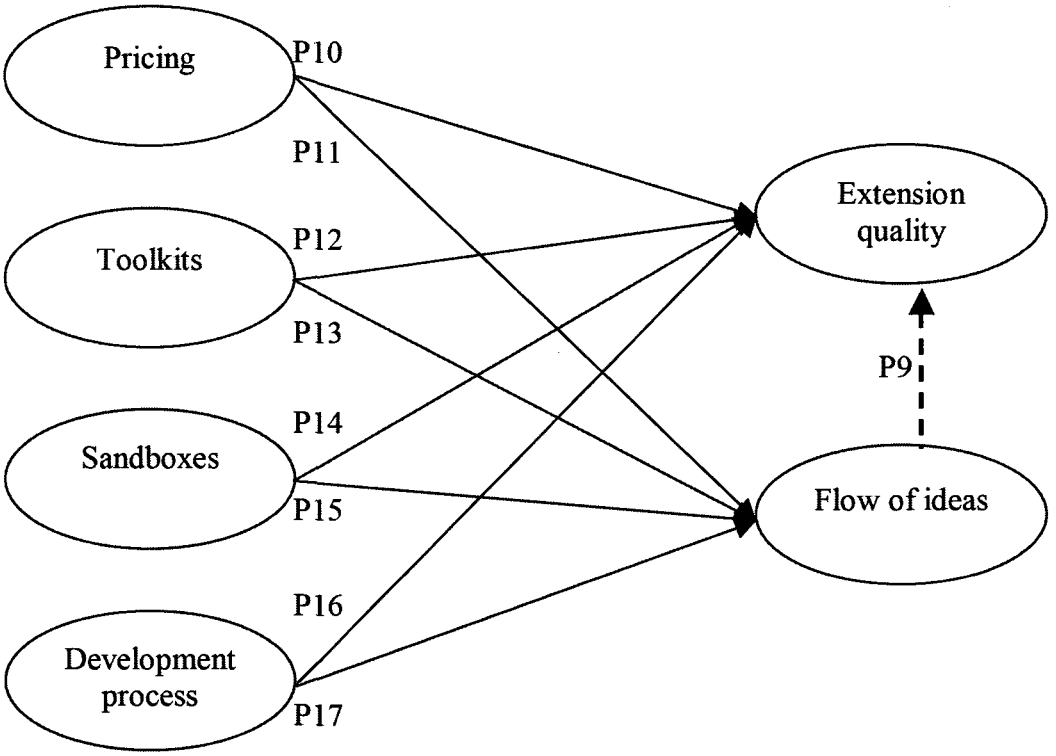

Figure 42. Framework for regulatory instruments

Table 35. List of regulatory instruments propositions

P10:	*Pricing is positively associated with extension quality.*
P11:	*Pricing is negatively associated with the flow of ideas.*
P12:	*Toolkits are positively associated with extension quality.*
P13:	*Toolkits are positively associated with the flow of ideas.*
P14:	*Sandboxes are positively associated with extension quality.*
P15:	*Sandboxes are positively associated with flow of ideas.*
P16:	*Development process is positively associated with extension quality. Established development process increases extensions quality (e.g. incubation process)*
P17:	*Development process is positively associated or negatively with flow of ideas depending on governance model.*

7 CONCLUSIONS, LIMITATIONS, AND FUTURE RESEARCH

7.1 Conclusions

In recent years, the quality management of platform complements had become growing concern for platform owners, as the value of a platform is realized as the value of the system comprising platform and complements. In our research we examined the different strategies adopted by platform owners to manage the quality of platform extensions that are developed by external parties.

This research contributes to the understanding of the strategies followed by platform owners to manage external innovation in the case of platform extensions in two areas: i) governance models, and ii) regulatory instruments.

The research key findings indicate that there are at least three governance models used by platform owners in conjunction with regulatory instrument to manage participation in the platform network of extension developers and control the quality for platform extensions. Each model is associated with non-trivial trade-offs in terms of governance, openness, flow of ideas, and quality of extensions.

Also there are different lever points at which the platform owners could manage participation and control the quality of extrusions such as: extension development (at the unit level), product configuration or deployment (at the integration level), and replication of the software and hardware ecosystem in which the extensions will be integrated (at the ecosystem level).

Another key finding was the case of a governance structure which is a hybrid of hierarchical and flat governance structures. The original framework that was suggested by Pisano & Verganti (2008) did not include such a category, leading them to classify both Linux and Apple's iPhone as examples of innovation communities, although they are very different in nature. In the case of the hybrid governance structure, while problems are decided on by the community, solutions are selected by the community and refined by the platform owner.

Also different types of governance and openness may be applied to different types of extensions. Internal extensions could be more tightly controlled than external extensions because internal extensions quality has a significant impact on the quality of the platform and on each other; a reduced flow of new ideas is traded off against higher quality. But in the case of external extensions, it is more important to allow new ideas to develop than to monitor their quality.

Finally, non- pricing regulatory instrument such as toolkits and sandboxes are important assets for the platform owners to harness the innovation process and leverage controlling the extensions quality by allowing extensions to be tested under conditions that individual developers cannot replicate, and where network members can share their experiences with the platform.

7.2 Limitations

This study has four limitations. First, time was a factor to limit the research scope to one type of platform complements and that is extensions.

Second, the main source of information is data available publicly on the websites of the platform owners. Eisenhardt (1989) recommends using multiple data collection methods to provide a strong validation of the results.

Third, in our research we have collected the data and simultaneously classified them and created our own taxonomy for models used to manage the process of developing platform extensions. This process can create two issues considered as a limitation for the research: i) creating a taxonomy and classifying entities at the same time could lead to miss some parts of the data, and ii) ignore the previous literature on participatory projects except the Pisano and Verganti (2008) that was extended in this research.

Fourth, the longitudinal aspect of studying the cases can be considered as a limitation for the research where the researcher did not consider the maturity level of platform that could have been added more valuable information that could help platform owners to determine the appropriate model depending on the platform maturity.

7.3 Opportunities for future research

This section identifies three opportunities for future research. First, expand the current study with a larger number (40 or more) of cases to test the propositions and explore possibilities of repetitive occurrence of certain models over the other.

Second, expand the research scope to include other type of complements such as services, applications and others and conduct a cross-case analysis to investigate the differences between the governance models for each type of complements.

Third, conduct a longitudinal study that sheds the light on the evolution of the platform and the different models adopted at every stage, if any, which will help platform owners to decide the suitable model to manage the extension development process at different stages of the platform life span.

Fourth, provide a closer examination for the role of regulatory instruments used in conjunction with these models to leverage the flow of ideas in the platform network for example developing a sandbox as a prototype that can be empirically evaluated.

8 REFERENCES

Baldwin, C.Y., & Clark, K.B. (2000). Chapter 3: What is modularity. In C.Y. Baldwin & K.B. Clark (Eds), *Design rules: The power of modularity, volume 2*. MIT Press..

Baldwin, C.Y., & Clark, K.B. (2006). The architecture of participation: Does code architecture mitigate free riding in the open source development model?. *Management* Science, 52(7), 1116-1127.

Bernal, J. (2008). Chapter 1: Application architecture. In J. Bernal (Ed), *Application Architecture for WebSphere: A Practical Approach to Building WebSphere Applications*. IBM Press.

Bosch, J. (2009). From software product lines to software ecosystems. *Accepted working paper for the 13th International Software Product Lines Conference (SPLC 2009)*. San Francisco, CA. USA.

Bosch, J., & Bosch-Sijtsema, P. (2009). From integration to composition: on the impact of software product lines, global development, and ecosystems. *Journal of Systems and Software*, In Press, Corrected proof, Available online 7 July 2009.

Boudreau, K., & Hagiu, A. (2009). Platform rules: multi-sided platforms as regulators. In A. Gawer (Ed.), *Platforms, markets and innovation*. Cheltenham, UK and Northampton, US: Edward Elgar, forthcoming.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture: A system of patterns*. Wiley.

Chen, W. J., Kirstein, H., Krook, D., Nair, K. H., & Pietrzak, P. (2006). Chapter 1: HTTP servers. In W.J. Chen, H. Kirstein, D. Krook, K. H. Nair, & P. Pietrzak (Eds), *Developing PHP applications for IBM data servers*. IBM Redbooks.

Chesbrough, H. (2003). The era of open innovation. *MIT Sloan Management Review, 44(3)*, 35–41.

Cusumano, M., & Gawer, A. (2002). The elements of platform leadership. *MIT Sloan Management Review, 43(3)*, 51-58.

Daum, B. (2004). Chapter 11: Developing Plug-ins for the Eclipse Platform. In B. Daum (Ed), *Professional Eclipse 3 for Java developers*. John Wiley & Sons.

Duenas, J., Parada, H., Cuadrado, F., Santillan, M., & Ruiz, J. (2007). Apache and Eclipse: Comparing open source incubators. *IEEE Software, 24(6)*, 90-98.

Dupuis, C. (2009). SpringSource tool suite now free. In *SpringSource team Blog*. Retrieved on June 10, 2009, from http://blog.springsource.com/2009/05/07/springsource-tool-suite-now-free/.

Eisenhardt, K. (1989). Building theories from case study research. *Academy of Management Review, 14(4)*, 532-550.

Emura, H., Shimodo, H., & Gomibuchi, T. (2007). *Firefox add-ons developer guide.* Retrieved on June, 24, 2009, from https://developer.mozilla.org/En/Firefox_addons_developer_guide.

Fleury, M., Stark, S., & Richards, N., JBoss Inc. (2005). *JBoss 4.0 the official guide.* Sams

Gawer, A., & Cusumano, M.A. (2008). How Companies Become Platform Providers. *MIT Sloan Management Review, 49(2)*, 28-35.

Gawer, A., & Henderson, R. (2007). Platform owner entry and innovation in complementary markets: Evidence from Intel. *Journal of Economics & Management Strategy, 16 (1)*, 1-34.

Gruber, O., Hargrave, B.J., McAffer, J., Rapicault, P., & Watson, T. (2005). The Eclipse 3.0 platform: Adopting OSGi technology. *IBM Systems Journal, 44(2),* 289-299.

Hagel III, J., Brown, S.J., & Davison, L. (2008). Shaping strategy in a world of constant disruption. *Harvard Business Review, October,* 81-89.

Hagiu, A. (2008). Two-Sided platforms: Product variety and pricing structures. *Journal of Economics & Management Strategy, 18(4).*

Hagiu, A. (2009). Quantity vs. quality and exclusion by two-sided platforms. *Harvard Business School, working paper.* Retrieved on June 4, 2009, from http://hbswk.hbs.edu/item/6182.html.

Hagiu, A., & Yoffie, D. (2009). What's your Google strategy?. *Harvard Business Review, 87(4),* 74-81.

Hemrajani, A. (2006a). Chapter 6: Overview of the Spring Framework. In A. Hemrajani (Ed) *Agile Java development with Spring, Hibernate, and Eclipse.* Sams.

Hemrajani, A. (2006b). Chapter 8: The Eclipse phenomenon. In A. Hemrajani (Ed) *Agile Java development with Spring, Hibernate, and Eclipse.* Sams.

Henderson, R.M., & Clark, K.B. (1990). Architectural Innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly, 35(1),* 9-20.

Iansiti, M., & Levien, R. (2004). Strategy as ecology. *Harvard Business Review, 82(3),* 68-78.

Iyer, B., & Davenport. T.H (2008). Reverse engineering Google's innovation machine. *Harvard Business Review, April,* 59-68.

Iyer, B., Lee, C.H., & Venkatraman, N. (2006). Managing in a "small world ecosystem": Lessons from the software sector. *California Management Review, 48(3),* 28-47.

Jamae, J., & Johnson, P. (2009). Chapter 2: Managing the JBoss Application Server. In J. Jamae, & P. Johnson (Eds), *JBoss in action: Configuring the JBoss application server.* Manning Publication.

Jeppesen, L.B. (2005). User toolkits for innovation: Consumers support each other. *Journal of Product Innovation Management 22(4),* 347-362.

Johner, H. (1999). Chapter 2: Building Blocks. In H. Johner (Ed), *IBM HTTP Server powered by Apache on RS/6000.* IBM Redbooks.

Keen M., Agopyan, A., Huebler, H., Puah, T., Schulze, T., & Soler Vilageliu, D. (2009). Chapter 1: Introduction to WebSphere application server V7.0. In M. Keen, A. Agopyan, H. Huebler, T. Puah, T. Schulze, & D. Soler Vilageliu (Eds), *WebSphere application server V7: Concept, planning an design.* IBM Press.

Kew,N. (2007). Chapter 2: The Apache Platform and Architecture. In The Apache Modules Book: Application development with Apache. Prentice Hall.

Knoernschild, K. (2008). WebSphere 7 and OSGi. In *Application platform strategy Blog*. Retrieved on June 25, 2009, from http://apsblog.burtongroup.com/2008/11/websphere-7-osgi.html.

Koufteros, X., Vonderembse, M., & Jayaram, J. (2005). Internal and external integration for product development: The contingency effects of uncertain, equivocality, and platform strategy. *Decision Science, 36 (1),* 97-133.

Krill, P. (2009). IBM puts the OSGi in WebSphere, WebSphere application server will be further modularized IBM says. In *InfoWorld*. Retrieved on June 25, 2009, from http://www.javaworld.com/javaworld/jw-03-2009/jw-03-osgi-in-websphere.html.

Kruckenberg, M., & Pipes, J. (2005). Chapter 4: MySQL System Architecture. In M. Kruckenberg, & J. Pipes (Eds), *Pro MySQL*. Apress.

Herzog, P. (2008). Innovation and the open innovation concept. In P. Herzog (Ed), *Open and closed innovation, different cultures for different strategies* (23). Gabler edition wissenschaft.

Lopez, D., & Blanco, J. (2006). Chapter 11: Multi processing and protocol modules: The evolution of Apache's architecture. In D. Lopez, & J. Blanco (Eds), *Apache Phrasebook*. Sams.

Machacek, J., Vukotic, A., Chakraborty, A., & Ditt, J. (2008). *Pro Spring 2.5*. Apress.

Maula, M., Keil, T., & Salmenkaita, J. P. (2006). Chapter 12: Open innovation in systematic innovation contexts. In H. Chesbrough, W. Vanhaverbeke, & J. West (Eds), *Open Innovation: Researching a New Paradigm* (249-257). Oxford University Press.

Messerschmitt, D.G., & Szyperski, C. (2003). Chapter 4: Creating Software. In D. G. Messerschmitt, & C. Szyperski (Eds), *Software ecosystems: Understanding an indispensible technology and industry*. The MIT Press.

Monson-Haefel, R, & Burke, B. (2006). *Enterprise JavaBeans 3.0* (5th ed.). O'Rielly Media, Inc.

Moore, J. (2006). Business ecosystems and the view from the firm. *The Antitrust Bulletin, 51(1),* 31-75.

Noll, J. (2007). Innovation in open source software development: A tale of two features. In J. Feller, B. Fitzgerald, W. Scacchi, A. Sillitti (Eds.), *Open source development, adoption and innovation* (109-120), IFIP/Springer.

Noori, N., & Weiss, M. (2009). Managing the quality of platform extensions. *Presented at the 3rd FLOSS International Workshop on Free / Libre Open Source Software,* Padua, Italy, 2009.

Pisano, G., & Verganti, R. (2008). Which kind of collaboration is right for you?. *Harvard Business Review, 86(12),* 78-86.

Prügl, R., & Schreier, M. (2006). Learning from leading-edge customers at The Sims: Opening up the innovation process using toolkits. *R&D Management, 36(3),* 237-250.

Ramme, K. (2005). Announcement Uno Runtime Environment (URE). *OpenOffice.org Newsletter.* Retrieved on July 22, 2009, from, http://www.mail-archive.com/announce@openoffice.org/msg00017.html.

Ridruejo, D. L, & Kallen, I. (2002). *Sams teach yourself Apache 2 in 24 hours*. Sams.

des Rivieres, J., & Wiegand, J. (2004). Eclipse: A platform for integrating development tools. *IBM Systems Journal, 43(2),* 371-383.

Romahn, U., (2008). *SpringSource releases SpringSource application platform.* Retrieved on June 10, 2009 from http://www.optaros.com/blogs/springsource-releases-springsource-application-platform.

Rossi, A., & Zamarian, M. (2006). Designing, producing and using artifacts in the structuration of firm knowledge: Evidence from proprietary and open processes of software development. *University of Trento, Quaderno DISA, 116.*

Sadtler, C., Albertoni, F., Fagalde, B., Kleinubing, T., Sjostrand, H., & Worland, K. (2006). Chapter 1: Introduction to WebSphere Application Server V6.1. In C. Sadtler,

F. Albertoni, B. Fagalde, T. Kleinubing, H. Sjostrand, & K. Worland (Ed), *WebSphere application server V6.1: planning and design*. IBM Press.

Schmidt, J. (2006). OpenOffice.org Extensions Infrastructure, What is it-What it can-What is planned. In *OpenOffice.org Conference 2006*. Retrieved on June 17, 2009, from http://marketing.openoffice.org/ooocon2006/presentations/wednesday_d10.pdf.

Shibuya, B., & Tamai, T. (2009). Understanding the Process of Participating in Open Source Communities. In *International Conference on Software Engineering*. Proceedings of the 2009 International Conference on Software Engineering (ICSE) Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development.

Shuen, A. (2008). *Web 2.0: A strategy guide*. O'Reilly.

Spinellis, D., & Gousios, G. (2009a). Chapter 12: When the Bazaar Sets Out to Build Cathedrals. In D. Spinellis, G. & Gousios (Eds), *Beautiful Architecture* (1st ed). O'Reilly Media, Inc.

Spinellis, D., Gousios, G. (2009b). Chapter 11: GNU Emacs: Creeping Featurism Is a Strength. In D. Spinellis, G. & Gousios (Eds), *Beautiful Architecture* (1st ed). O'Reilly Media, Inc.

Stein, L., & MacEachern, D. (1999). Chapter 1: Server-Side Programming with Apache-The Apache Project. In L. Stein, & D. MacEachern (Es), *Writing Apache modules with Perl and C.* O'Reilly Media, Inc.

Suàrez-Potts, L. (2006). State of the oroject, year 6. In *OpenOffice.org Conference 2006.* Retrieved on June 19, 2009, from http://www.openoffice.org/editorial/state_of_the_project_year_6.html.

Sun Microsystems, Inc. (2008). Chapter 4: Extensibility and Developers Features. In *StarOffice™ 9 Office Suite, Guide to New Features.* Retrieved on August 30, 2009, from www.sun.com/software/staroffice/docs/StarOffice9_WhatsNew.pdf.

Thomke, S., & von Hippel, E. (2002). Customers as innovators: A new way to create value. *Harvard Business Review, 80(4),* 74-81.

Tuomi, I. (2003). *Networks of Innovation: Change and meaning in the age of the internet.* Oxford University Press.

Venugopalan, V. (2005). Chapter 4: Developers sandbox. In *CVS best practices.* Retrieved on July16, 2009, from, http://tldp.org/REF/CVS-BestPractices/html/section1-devsandbox.html.

von Hippel, E. (2001). PERSPECTIVE: User toolkits for innovation. *The Journal of Product Innovation Management, 18(2001),* 247-257.

von Stamm, B. (2008). *Managing innovation, design, and creativity* (2nd ed.). Wiley.

Vujovic, S., & Ulhoi, J. P. (2008). Chapter 8: Opening up the Innovation Process, Different Organizational Strategies. In R. M. Burton, B.H. Eriksen, D.D. Håkonsson, T. Knudsen, & C.C. Snow (Eds.), *Designing Organizations, 21st century appraoches.* Springer US.

Walls, C., & Breidenbach, R. (2007). *Spring in Action* (2nd ed.). Manning Publications.

Watson, R.T., Wynn, D., & Boudreau, M.C. (2005). JBOSS: The evolution of professional open source software. *MIS Quarterly Executive, 4(3),* 329-341.

Watson, R.T., Wynn, D., Boudreau, M.C, York, P.T., Greiner, M.E., & Wynn, D. (2008). The business of open source: Tracking the changing competitive conditions of the software industry. *Communications of the ACM, 51 (4),* 41-46.

West, J., & Gallagher, S. (2006). Challenges of open innovation: the paradox of firm investment in open-source software. *R&D Management. 36(3),* 319-331.

West, J., & O'Mahony, S. (2008). The role of participation architecture in growing sponsored open source communities. *Industry and Innovation, 15 (2),* 145-168.

Yeow, C. (2005). Chapter 1: Introducing Your New Favorite Web Browser. In C. Yeow (Ed), *Firefox secrets.* SitePoint.