

# Managing the Quality of Platform Extensions

Nadia Noori and Michael Weiss

Department of Systems and Computer Engineering

Carleton University, Ottawa, Canada

## Introduction

Innovation increasingly happens in networks (Tuomi, 2002) or ecosystems (Moore, 2006); few companies can still afford to innovate on their own. The question is no longer whether or not to collaborate, but how to best leverage a network of external parties (Pisano & Verganti 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 or ecosystem is a focal organization or keystone (Iansiti and Levien, 2004) that provides a platform which allows the parties in the network to contribute. Here, we define a platform with Haigu et al. (2009) as a product, service or technology that provides a foundation for other parties to develop complementary products. This platform can be provided 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).

In this paper we explore an issue that has not received sufficient attention by the research literature, although it is of great practical importance. We examine the strategies followed by platform providers to manage the quality of platform complements developed by other parties. To keep the study to a manageable size, we limit our scope to software platforms and to one type of complement: platform extensions, which extend the functionality of a platform beyond its core capabilities. Quality management of platform extensions is a concern for platform providers, as the value of a platform is realized as the value of the system comprising platform and complements.

The audience for our research includes potential platform providers who need to understand what strategies other platform providers have adopted and for why, third party developers of

extensions who need to know about tools and me to harness platforms, and researchers who are provided with new research opportunities to develop models for managing external innovation in platforms.

This paper is organized as follows. First we provide a review of related work on platform strategies, open innovation, and toolkits. Then we describe our case-based research method. Next, we present the initial findings from our first round of cases examined. In particular, we examine the features and quality management process of two open source platforms, Mozilla Firefox and the Eclipse platform. We conclude with our contributions and areas for future research.

## **Related work**

Most products today are systemic. A platform, thus, only has value to users, if there is a sufficient number of high quality complements (extensions) available. This is why some platform providers choose to impose tight controls on who can add what types of extensions. However, when platform providers impose controls that are too restrictive, this can also limit the level of participation by other parties, that is, restrict the influx of new product ideas to the platform provider.

Platform providers can also offer toolkits that allow external parties to perform some of the tasks related to quality management on their own. Toolkit encode design rules that the extensions will work properly. Shifting responsibilities for quality management to other parties helps accelerate the development process and leads to greater product quality (Thomke & von Hippel, 2002).

We have divided the literature into three streams: platform strategies, open innovation, and toolkits.

## **Platform strategies**

Innovations have become increasingly systemic as identified by Maula et al. (2006). Focal companies define the architecture of the systemic innovation. Yet, they depend on external parties to provide necessary components that complement the core innovation (the platform) by the focal company. These external parties are not under the direct control of the focal company. To tap into external resources, the focal company must demonstrate a credible

commitment to the systemic innovation instead. This means that there must be significant benefits to both parties.

Iyer & Davenport (2008) describe the innovation ecosystem Google has built around a proprietary platform. At the core of Google's ecosystem is its computing infrastructure built to support its search operations. It enables Google to leverage third-party innovation while maintaining architectural control. The infrastructure comprises services such as Google Ajax Search, AdSense, and Maps. Third parties can create mashups that incorporate these services. They also get exposure to Google's huge user base. In exchange, the Google gains access to valuable analytics, and to ideas for new applications and improvements to its services than it could not have discovered internally.

### **Open innovation**

**Chesbrough (2003)**: "the boundary between a firm and its surrounding environment is more porous, enabling innovation to move easily between the two".

Pisano & Verganti (2008) explore how innovation leaders leverage 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 a flow of information between the platform provider and external parties. Governance determines who sets the direction of innovation and who appropriates the value created from the innovation. These distinctions result in four basic modes of collaboration with distinct trade-offs: elite circle (closed, hierarchical), consortium (closed, flat), innovation mall (open, hierarchical) and innovation community (open, flat).

### **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, which allows users to carry out some of the product development tasks themselves.

Mediating toolkits also play a significant role in ensuring quality as identified by Rossi & Zamarian (2006) and Thomke & von Hippel (2002).

## **Lessons learned**

Collaborative innovation is becoming a staple of modern business practices. 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. Open innovation leads to porous boundaries between firms, which facilitates information flow with external parties, but also results in a loss of direct control over the quality of product features.

## **Research method**

Because of the novelty of the research area, we adopted a case study approach (Eisenhardt, 1989). For each case, we identify the core features of the models for quality management adopted by the platform provider, and group them into strategies for enabling external innovation among extension developers through a cross-case analysis. To date we have completed two of four cases in the first round of analysis. This will be followed by another round of collecting cases and analyzing them if any new patterns emerge. Finally, we expect to develop propositions from these features that define the relationship between platforms and extensions in terms of control, quality, and flow of ideas.

Our unit of analysis is an extensible software platform that was published between 2004 and 2009. For each platform, we build a case profile, which includes information on platform architecture, open/closed nature of platform, development tools, deployment methods, testing strategies, number of external parties and extensions published, and community participation architecture. As part of the first round of cases collected, we examined how two open source platforms, the Mozilla Firefox project and the Eclipse project, manage the quality of platform extensions.

## Findings

Platform providers are concerned with two types of quality issues: the quality of individual extensions, and the integrity of the platform. Quality assurance of individual extension is analogous to unit testing, and is focused on the extension in isolation. Ensuring integrity is analogous to integration testing: while all extensions may demonstrated the expected functionality in isolation, they may not work with one another, or may even break functionality in the platform core. We will argue that current models for platform quality management support quality assurance of individual extensions, but do not, at present, support assurance of platform integrity at the extension level. The only examples we are aware of that ensure platform integrity are at the level of applications that build on the platform. Apple's incubation process for iPhone applications is an example.

### Platform extensions

A platform consists of a platform core, which provides common functionality, and platform extensions, which implement additional functionality. Platform extensions may have different forms and be integrated at different levels within the platform as shown in Figure 1. The first distinction is between external and internal extensions. The second distinction is between internal extensions that are true extensions and those that have been incorporated into the platform core.

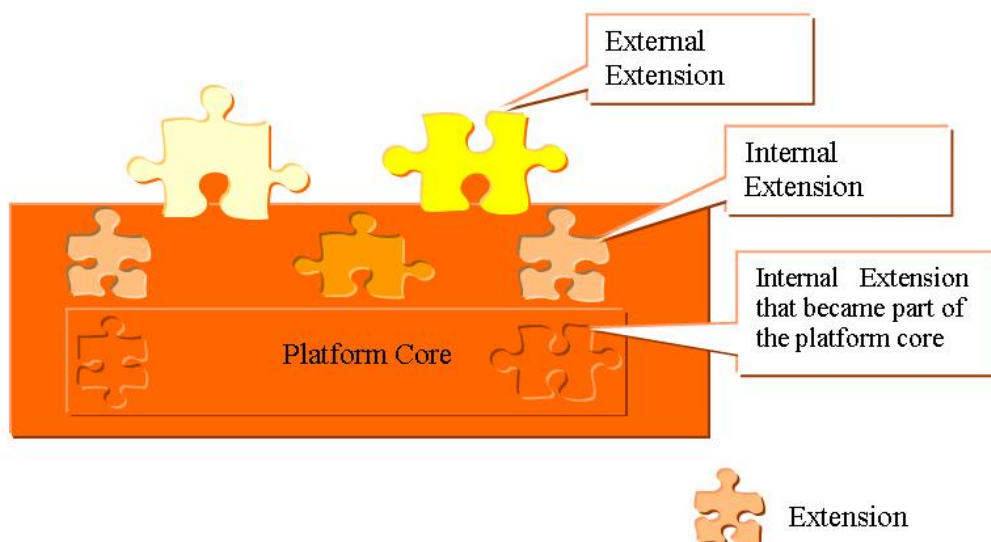


Figure 1. Types of platform extensions

The distinction between internal and external extensions originates with the microkernel architecture implemented by systems that need to be made available in multiple configurations which share a common core functionality (Buschmann et al., 1996). A microkernel architecture has a minimal runtime (corresponding to the platform core) that provides hooks for adding extensions. All required functionality is provided in the form of internal extensions that are integrated with the platform, while application-specific functionality may be added through external extensions.

Note that extension status can change from external to internal or from internal extension to fully integrated with the core, as a platform evolves. This transmission is not random, but usually follows a process controlled by the platform providers for managing extension development. One example is the Tabbed Browsing feature of the Firefox web browser (discussed below), which was initially available as an external extension, but was later bundles as part of the Firefox core platform.

There are three possible scenarios for extending a platform that we expect to find in our case studies. As shown in Figure 2, a platform can (a) make use of internal extensions, but not allow other parties to freely add extensions, (b) have a monolithic core, and only support external extensions, or (c) allow both internal and external extensions. Platforms without extensions are outside our scope. In our first round of case studies, we identified the Eclipse platform as an example of scenario (c), and the Firefox web browser as a hybrid between scenarios (b) and (c). Both are described in more detail below. Case (a) is illustrated by the Spring Extensions initiative for extending the Spring application server framework.<sup>1</sup> As we have just started exploring this case, we did not include it here.

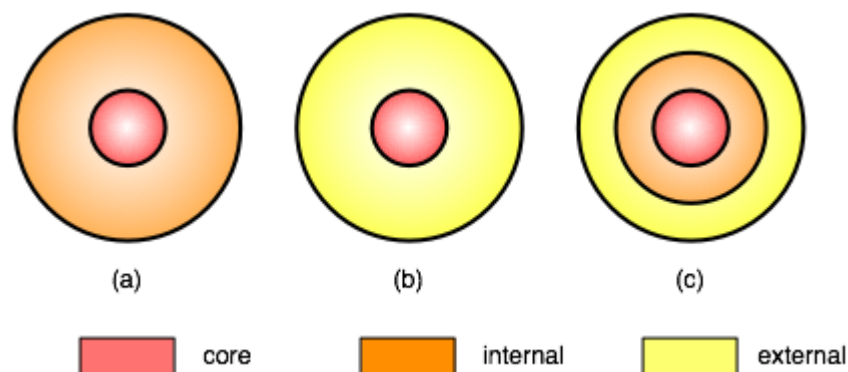


Figure 2. Platform extensibility scenarios

---

1. <http://www.springsource.org/extensions/about>

## Quality management

Platform providers have different levers to manage the quality of platform extensions. Some providers only allow selected parties (such as licensed partners or internal developers) to work on the platform core, but anyone else can create applications using the platform. An example of this is application development for Microsoft Windows. This type of control over platform extensions is outside the scope of our study. Other providers allow the development of platform extensions, including extensions to the platform core, but require these extensions to undergo a process that controls their deployment. An example is development of extensions for Eclipse core projects.

We refine the Pisano & Verganti (2008) framework to adapt it to quality management. Our framework has three dimensions: governance, openness, and flow of information.

1. The governance structure of a network can be either hierarchical, flat, or a hybrid between these extremes. In a hierarchical governance structure, the platform provider 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 provider.<sup>2</sup>
2. Openness of an innovation network refers to the degree to which participation in the network is open. In an open network, any party (be they partners, customers, or even competitors) can contribute. Open source projects are examples of this type of network. In a closed network, the focal company or sponsor of the network selects who can participate based on the capabilities and resources required for the innovation.
3. Flow of information decides on the extent and richness of the information exchanged through the network. Restricted information flow 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

---

2. 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 flat governance, which are clearly different types of platforms.

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. Instead, the fully specified problem is posted to the network.

## **Mozilla Firefox project**

Firefox is part of the Mozilla project and its goal is to develop an open source web browser. It is considered one the best web browsers and has gained significant marketshare in recent years.<sup>3</sup>

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.

Firefox allows third parties to develop extensions that enhance its functionality. An extension could be a minor change such as adding a button to the toolbar, or a major change such as a whole new feature.<sup>4</sup> The user community has created many Firefox extensions that serve specific needs, which are not provided by the core platform, such as ad blockers, annotation tools, or bookmark managers. Different from the extensions we study here, Firefox also supports "plugins" which are really helper applications for displaying content in formats that are not natively supported by the browser (for example the Adobe Reader plugin for displaying PDF documents inside the browser).<sup>5</sup>

Most Firefox extensions are created in an open manner, and defined and selected by the community without involvement of the platform provider. In Table 1, these are categorized as 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

---

3. Firefox Market share, <http://marketshare.hitslink.com/firefox-market-share.aspx> , Retrieved on April 18, 2009.

4. Firefox Extensions, <https://developer.mozilla.org/en/Extensions>, Retrieved on April 19, 2009.

5. Firefox, Plugins, <https://developer.mozilla.org/en/Extensions> , Retrieved on April 19, 2009

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.

	Hierarchical	Hierarchical/Flat	Flat
Open		Firefox core extension "Internal"	Firefox extension External
Closed	Commercial extension External		

Table 1. Collaboration modes for Firefox extensions

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.<sup>6</sup> The governance structure for a commercial extension developed by a single company is hierarchical. This accounts for the third entry in Table 1.

Figure 3 shows the community participation structure for the Mozilla Firefox platform. It shows 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 participation architecture in Figure 3. The "outer circle" represents the developers of all other, that is, external Firefox extensions.

---

6. This and other commercial Firefox extensions are described in <http://www.slideshare.net/glazou/monetizing-firefox-extensions>.

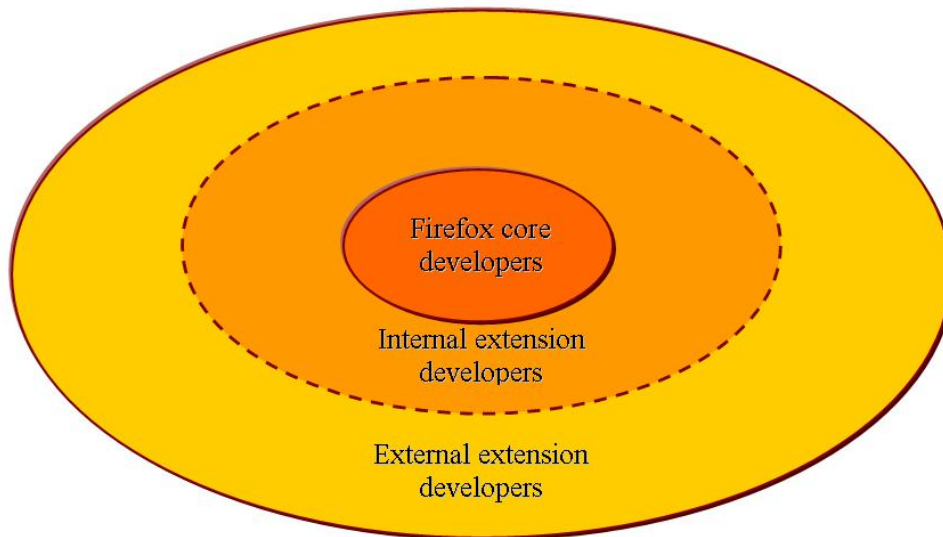


Figure 3. Mozilla Firefox community participation architecture

### **Eclipse project**

The Eclipse platform has evolved from an integration platform for tools (des Rievieres & Weigand, 2004) to a general platform for integrating applications and services (Gruber et al., 2005). Historically, it is interesting to note that, like Firefox, it also evolved from a commercial product that was subsequently released as an open source project. Unlike Firefox, however, it has "stuck" to its commercial roots more closely. Where participation in the Firefox project can best be described as a meritocracy, the governance structure of the Eclipse project is driven by its member organizations.

Eclipse is designed to be highly extensible. At its core is a minimal runtime that provides tools for extension management. All functionality of Eclipse (even "core" functionality such as basic UI elements) is implemented in the form of extensions, which are referred to "plug-ins" in the Eclipse environment. Plug-ins are the basic distribution unit of functionality in Eclipse. Each plug-in (plug-declares its dependencies on other plug-ins. A plug-in declares extension points, and implements extensions to the extension points of other plug-ins (des Rievieres & Weigand, 2004).

With Eclipse 3.0, the Eclipse platform core has been reimplemented as an OSGi-compatible service container, Eclipse Equinox (Gruber et al. 2005). Its Component Oriented Development and Assembly (CODA) component model separates the development of software components from their customization and assembly. It supports the dynamic

configuration of systems.

The Eclipse project is organized as a set of core projects, which are featured prominently on the Eclipse website and have to undergo a strict incubation process in order to be accepted. (Duenas et al, 2007). Each project is implemented as a set of plug-ins. They are the equivalent of internal extensions. Regular plug-ins do not require incubation, and are listed on repositories such as Eclipse Plugin Central. 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 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 2 categorizes the different types of extensions of the Eclipse platform using our framework.

	Hierarchical	Hierarchical/Flat	Flat
Open		Eclipse core project Internal	Eclipse plug-in External
Closed	Commercial plug-in External		

Table 2. Collaboration modes for Eclipse extensions

Figure 4 shows the project community participation architecture of Eclipse. Here the delineation between internal and external extensions (core projects and plug-ins, respectively) is present by design in the governance structure. At the core are developers of the Eclipse runtime (core). The inner circle consists of core project developers, and the outer circle of plug-in developers.

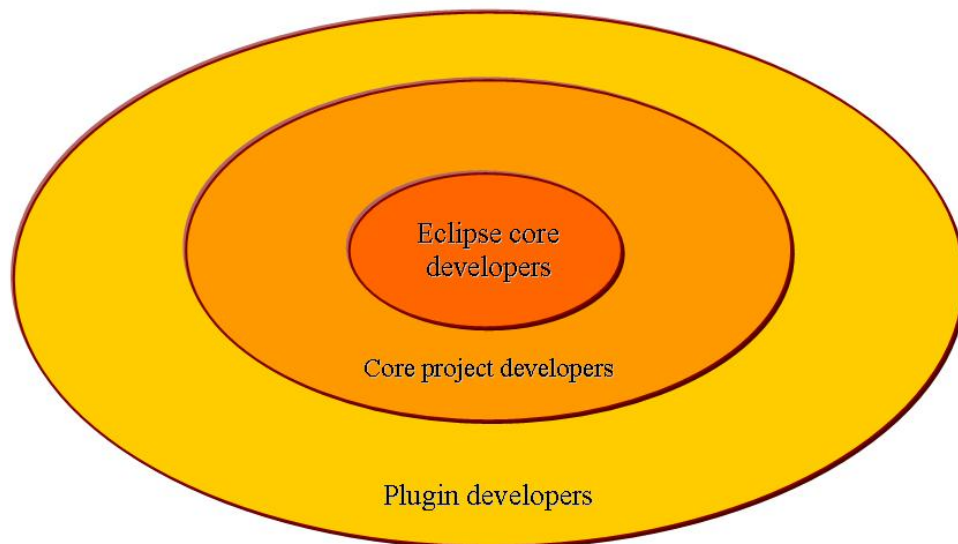


Figure 4. Eclipse project community participation architecture

## Toolkits

### Toolkit, third-party toolkit, and sandbox

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.<sup>7</sup>

### Flow of information

## Conclusion

---

7. The Home of Mozilla QA, <http://quality.mozilla.org/test>, Retrieved on April 18, 2009

There is no single approach that platform providers can follow to manage the quality of external innovation. Each approach is associated with complex trade-offs in terms of governance, openness, and flow of ideas. There are also different lever points at which control can be exercised: extension development, product configuration or deployment, and platform development.

Framework for managing quality of extensions ...

Sandbox concept: sandbox to be used when complexity of product configuration or deployment is high or requires special hardware, and when there are multiple users ...

Although we have limited our attention to platform extensions, we believe that our findings also have implications for managing the quality of other types of complements. In future work, we plan to pursue some of the paths not taken in this study. For example, we want to examine the quality management of applications that build on top of a platform such as the iPhone applications in Apple's AppStore. Another venue for future work is to develop the sandbox concept further by implementing a working prototype and applying it to an existing platform.

## References

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley.

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

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

Duenas, J., Parada, H., Cuadrado, F., Santillan, M., & Ruiz, J. (2007), Apache and Eclipse: Comparing opensource incubators, *IEEE Software*, November/December, 90-98.

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.

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

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

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

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. *Open Source Development, Adoption and Innovation*, IFIP/Springer, 109-120.

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.

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

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.

Shuen, A. (2008). *Web 2.0: A Strategy Guide*. O'Reilly.

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.

von Stamm, B. (2008). *Managing Innovation, Design, and Creativity*. Second edition. Wiley.

