How Quality Attributes of Software Platform Architectures Influence Software Ecosystems

Slinger Jansen
Utrecht University, the Netherlands
slinger@slingerjansen.nl

ABSTRACT

When developing commercial applications, developers seldomly start from scratch. Generally, they use software platforms and extend them, joining an ever growing software ecosystem surrounding the platform. In this paper, the relationships between architecture and platform adoption are explored by analyzing the results of interviews and document study of five case studies of platform extenders. It is found that platform architecture plays a minor role in platform adoption by platform extenders, but that quality attributes strongly influence an architect's design choices when extending a platform. The findings of this work can be used by platform developers to improve platform extendibility and usability.

Categories and Subject Descriptors

K.6.3 [**Software Engineering**]: Software development; D.2.11 [**Software Architecture**]: Domain-specific architectures

General Terms

Design

Keywords

Software platforms, platform architecture, software ecosystems

1. SOFTWARE PLATFORMS

The best example of successful software reuse is found in ecosystem based software platforms [1, 2, 3]. Ecosystem based software platforms receive little attention from the scientific community. This is surprising, since software application developers generally start development by choosing a software platform that already satisfies a large part of the required functionality by the developer's customers. Platform based development differs from traditional development in that software architecture and platform reuse mechanisms have a central role in the development of platform extensions. There is little knowledge available, however, on how a successful and easy-to-adopt software platform architecture can be developed. This paper aims to provide insight into software platforms and platform usability for platform developers and for third-party developers (i.e., platform extenders). Furthermore, the relation between platform adoptability by platform extenders and a platform's architecture is studied.

An **ecosystem based software platform** is defined as a collection of software artifacts that form a coherent whole on which applications can be built by third parties. Typically, software platforms consist of an architecture, a software stack, programming languages, related software components, and a graphical user interface. Platforms can be fundamental technology platforms, such as a web platform, whereas other platforms are functionally oriented. An example of a technology platform is Ruby on Rails, which provides extenders with functionality to create any type of web application. An example of a platform that is more functionally oriented is Microsoft CRM (MSCRM), which is used by platform extenders because of its functional characteristics, i.e., the CRM functionality that is already present in the platform. The iPhone and SAP platforms are hybrids: both the technological platform and the functionality of the platform are relevant to third-party developers.

A **software ecosystem (SECO)** is a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts [4]. The success of software vendors that employ software ecosystems ([5, 6]) to generate activity around their software platforms and services is undeniable. Iyer et al. [7] state that *"the focus is not longer about competition between products, but about competition between platforms. ... [participants] compete not just on the technical features that differentiate their consoles but also on their ability to orchestrate their network of third-party support."*

Two types of entities are relevant in regards to platforms: platform suppliers and extenders. A **platform supplier** is an organization that maintains and supports a platform and that actively stimulates the use and adoption of it. A **platform extender** is an organization that adopts and develops on top of the platform to increase the speed at which it can develop software, with the aim of supplying a specific solution to a problem. Platform extenders are also known as plug-in builders and value added resellers. Software architecture must influence how platform extenders use a software platform to build domain specific solutions [8]. After all, the architectural decision to keep an architecture closed or to open it up for all, influences the degree of freedom for any platform extender [9].

This paper presents case studies of platform extenders and their experiences with and motivations for using a platform. The research approach is presented in Section 2, along with two hypothesis that aim to explore the relationships between architecture and platform adoption. The platforms under study and case studies

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

Copyright is held by the author/owner(s). Publication rights licensed to ACM.

WEA'13, August 19, 2013, Saint Petersburg, Russia
ACM 978-1-4503-2314-7/13/08
http://dx.doi.org/10.1145/2501585.2501587

themselves are presented in Sections 3 (Ruby on Rails), 4 (iPhone), 5 (SAP), and 6 (Microsoft CRM). In Section 7 the hypothesis that platform extenders are strongly influenced by a platform's architecture is refuted, and the hypothesis that developer behavior is strongly influenced is accepted, based on a number of platform architecture quality attributes. Finally, in Section 8, the findings are summarized and show future directions into standardization of API documentation, platform adoption, and change propagation throughout a platform ecosystem.

2. RESEARCH APPROACH

The relationships between platform adoption and platform architecture are explored through the following hypotheses:

- H_1**:** Platform extenders have major architecture concerns when selecting a platform
- H_2**:** The platform architecture influences the development behavior of platform extenders and influences the platform's supporting software ecosystem

For H_1 the motivations to start developing on a specific platform of several platform extenders were investigated. For H_2 four platforms and the impact of specific architectural decisions on platform extenders were investigated.

The platforms under study sprouted from one of two situations: (1) an Independent Software Vendor (ISV) had a product that was widely implemented and customized, to the extent that the ISV decided to open up specific parts of the interfaces through SDKs and APIs and started building up a software ecosystem or (2) an ISV developed a platform from the ground up with the motivation to have it adopted by many value added resellers. Examples of the former are Microsoft CRM (MSCRM) and SAP, which started out as monolithical products, but soon had to be extended by large numbers of third-parties and customers. Examples of the latter are the iPhone platform and the Ruby RoR platform, which were both specifically developed with a platform in mind.

Table 1 lists the platforms that have been studied. Platforms were selected with three criteria in mind: two platforms could not be similar (no two mobile platforms, for instance), there had to be many platform extenders available, and platform extenders use the platform to generate business around the created extensions. Each case study consists of two steps: first, a literature study was conducted into the platform by studying platform development reference documentation. Secondly, 90 minute interviews were conducted at software companies that extended the platform under study. When the platform extender was smaller than five people, the lead developer was interviewed in one session of 120 minutes. Case results were processed immediately after the interviews, as to avoid knowledge drain.

3. CASE STUDY 1: RUBY ON RAILS

Rails was created in 2003 by David Heinemeier Hansson and has since been extended by the Rails core team and several hundreds of open source contributors. Rails is a web application platform, written in the Ruby programming language. Rails is quickly becoming one of the most used platforms for web development, for two reasons. First, it emphasizes convention over configuration, resulting in an easy to deploy, easy to install, and easy to maintain development platform. Secondly, it emphasizes code reuse, to avoid code cloning. These two adages show that Rails is a developer's platform, more than anything else. There are several rich repositories of Ruby extensions, called gems, that are being reused extensively. These components follow the convention over configuration adage and are generally easy to deploy and test. Furthermore, through an interesting cooperation with Apple, Rails is shipped with any copy of the widespread Mac OS X operating system.

Case study 1 - *RoRExt1* is a two person company that builds web applications. Developers switched from PHP to RoR in 2006. The main reasons for switching were the productiveness boost, the ease of adoption, and development comfort. *"Development comfort is the main reason for any developer to switch platforms. In the RoR platform, the abundance of removed annoying problems compared to other platforms has convinced us of this."* Development comfort was increased by the availability of semi-automatic database migrations, scaffolding, and rapid prototyping. Furthermore, ecosystem specific aspects were mentioned, such as the openness of the platform, the availability of well-tested components, and the platform's stack independence. Learnability was increased by the availability of complete documentation. Finally, the platform is considered backward compatible over subsequent releases, which makes it easy to maintain older applications as well.

The developers at RoRExt1 praise the developers of the RoR platform and the Ruby language frequently. Some mechanisms that get special attention, such as the clean way in which a junior developer will never need to delve into the core code (for instance of the business object layer), the way in which one can stay up to date of database changes (using the active repository pattern), and the extensive way in which reflection is used throughout the language. A downside was identified by *RoRExt1* as well: no comprehensive IDE exists that includes code completion and application visualization. Furthermore, the platform is not supported by a debugger that can be used to debug applications, even though several of such tools are presently under development, for instance in the Eclipse framework.

Case study 2 - The company *RoRExt2* is a four-man company that has created several advanced websites using Ruby on Rails. The employees of the company got frustrated with PHP, which they used until 2006. The switch was easily made, mostly due to an active community surrounding the Ruby RoR platform. *RoRExt2* builds highly specialized web applications. These applications have different non-functional requirements, which means that different parts of the Ruby RoR platform need to be studied each time. For example, for a recent project, a large number of database transactions needed to be performed at high speed, which resulted in the building of an optimized data access layer, which circumvented the one available within the Rails platform.

The main reasons for *RoRExt2* to adopt RoR were that it was easy to learn, innovative, and most of all a boost in productivity. As soon as the developers at *RoRExt2* realised that their productivity increased by 50% in the first months of using RoR, all solutions from then on were built using the platform. A downside to the platform is its development stability: it is unclear to *RoRExt2* at what point different factions of developers will split up and develop their own clones and components for the platform.

4. CASE STUDY 2: IPHONE

The success of the iPhone platform is undeniable. It is considered one of the most successful examples of a platform that made it into the consumer's home. iPhone development is done using XCode, a development tool that only runs on a Mac, and the Cocoa Touch framework, with its Objective C language. iPhone OS is largely based on Apple's OS X operating system, which has enabled Apple to fairly quickly develop a reliable operating system for third-party applications. The ecosystem surrounding Apple and its iPhone consists of technology suppliers, application builders,

Table 1: Platforms Under Study

	MS CRM	iPhone	RoR	SAP
Programming language	.Net compatible such as C#, VB.net, etc.	Objective C	Ruby	ABAP, Java
Development environment	Visual Studio	XCode	TextMate	Netweaver Development Infrastrcture (Eclipse based), ABAP Workbench
SDK	Dynamics	iPhone SDK	Several	ABAP libraries
Hardware platforms	Any server	iPhone	Any server	Any server
Operating systems	Windows	Cocoa Touch iPhone OS	Mac, Windows, Linux	Windows, Unix
Delivery mechanisms	Application Assemblies	App Package	Ruby Gems	PAT files, through SAP's delivery mechanism

mobile operators, and Apple itself. Apple wishes to retain control on the ecosystem. To begin with, Apple does not allow applications to reuse functionality from other applications, whereas others such as Eclipse and Ruby on Rails do. Secondly, Apple has a strict (yet badly communicated) policy as to which applications are accepted to the App Store.

Case study - *iPhoneExt* is a two man company that develops iPhone applications. The company is a university start-up, and has created several large iPhone applications. These applications have been downloaded several thousand times.

iPhoneExt chose to start building for the iPhone for a number of reasons. The first professional App was built because of market pressure: customers specifically requested iPhone applications. The success of the iPhone itself stimulated activity around the device. *iPhoneExt* further states that the infrastructure for publishing and selling applications makes it easy to make money with an iPhone application. Furthermore, the Cocoa Touch framework's stability is mentioned as an important factor.

There are downsides to the iPhone platform. First and foremost, the platform is closed in that it does not allow developers to access all the available features. Secondly, the compiler should do more to support developers, by providing better feedback, suggesting weak spots in the code, and by improved error reporting. Thirdly, the system is relatively opaque, i.e., due to the fact that no internal APIs are documented it is hard for developers to make use of all functionality of the platform. Finally, the developers complain about having to do garbage collection themselves.

Several other comments are provided: the documentation does not cover certain design decisions, which are generally described in books about the platform. Furthermore, the use of application templates only covers a small part of the work: developers still need to build their specific solution. Finally, due to the newness of the platform, legacy and backwards compatibility hardly cause any problems yet. One elegant architectural solution for legacy problems, for instance with older hardware such as the iPod touch, is the use of a microkernel that makes sure how a hardware component should be called and what to do in case of version problems.

5. CASE STUDY 3: SAP

SAPERP is one of the oldest of ERP software platforms: an ERP platform created in 1972, that has generated enough revenue for SAP to grow to be the third largest software vendor in the world in terms of revenue. SAP's platform, which evolved through several generations, is now one of the largest all-encompassing ERP systems that can be employed by a business. SAP is well known for its complex but powerful system and some of its customers are the largest companies in the world.

At an early stage, SAP introduced the SAP developer community, a network of developers that cooperatively develops SAP components in its native programming language, ABAP (Advanced Business Application Programming language). Furthermore, due to the complex deployment and implementation processes, SAP started developing an ecosystem around its product consisting of service companies and third-party developers.

Case study - *SAPExt* is a Dutch company with 25 employees. *SAPExt* builds an ABAP application that assists management in the process of closing the financial year for large organizations using SAP. The *SAPExt* product creates a spreadsheet that is used by different stakeholders to monitor closing progress and to control and approve the closing of separate departments. *SAPExt* further facilitates, with checks and controls, that an organization meets the criteria for Sarbanes–Oxley compliancy, which are the regulations governing that every financial report in an organization needs to be signed off by the responsible employees, as to prevent fraud and incorrect reporting. *SAPExt* satisfies a need for large organizations that has not been provided by SAP in the past. *SAPExt* is an active partner of SAP. *SAPExt* code consists mainly of ABAP code, i.e., it runs on the lowest level of the SAP layers. A *SAPExt* application is delivered using SAP's delivery system. *SAPExt* has been created using mostly the data access layer to create custom entities and to read data from SAPERP. *SAPExt* only accesses transactions and does not require other business logic from SAPERP.

In regards to development *SAPExt* states that some of the SAPERP peculiars can not be learned from a book. This is not due to bad documentation, which is abundantly available from both SAP and its developer community, but more a legacy problem. Another SAP problem is that each project organization that implements SAPERP has its own naming conventions and implementation particulars. This lack of standardisation has created a challenge for *SAPExt* to implement its system. The SAPERP system has been very stable across subsequent releases, which is seen by *SAPExt* as an advantage over other platforms.

Finally, *SAPExt* is bound by the functionality offered by the SAPERP platform. Some problems it is currently experiencing are the limited user interface, limited APIs for development, slow transaction processing, limited documentation, and a constricting legacy development language. *SAPExt* is, on the other hand, balancing these problems against the gains coming from a healthy ecosystem, a large number of (potential) customers, and a robust well-tested and documented platform.

6. CASE STUDY 4: MSCRM

Microsoft Dynamics CRM is a Customer Relationship Management (CRM) software platform developed by Microsoft. It is a part of the Microsoft Dynamics family of business tools. MSCRM provides functionality that supports businesses in their sales and customer management process. It runs on a Microsoft stack, i.e., it requires Microsoft products such as databases and servers, before

Table 2: Platform Selection Motivation (greyed out cells are architecture related, resulting in the rejection of H_1)

	RoRExt1	RoRExt2	iPhoneExt	SAPExt	MSCRMExt
First	Technology driven	Technology driven	Market driven	Market driven	Available features
Second	Available features	Available features	Ecosystem size	Available features	Available developers
Third	Openness	Lively ecosystem	Framework stability	Legacy	Market driven

it can be used. MSCRM is currently one of the most popular CRM products in a multi-billion business arena.

The developing organization behind MSCRM does not strive to supply the CRM product with the most features, but instead supplies an extendible platform that, with development from the customer or a third-party, can provide a customized and fitting solution. One of MSCRM's advantages in the CRM market is that MSCRM couples with other Microsoft ERP products, such as the e-mail server application of Microsoft called Outlook.

Case study - The company *MSCRMExt* is an ERP vendor that sells a large product that extends the MSCRM platform. *MSCRMExt* currently has around 600 customers worldwide and 28 employees. The product is sold through resellers and through an internal sales department. *MSCRMExt*'s product works besides MSCRM, which means that the products can function independently from each other, even though customers will generally not be aware of that. Interaction with MSCRM is initiated in different ways. Regularly, data objects that are shared are synchronized, such that parts of the data in the MSCRM database are the same as the data in *MSCRMExt*'s product's database. This data is only used when for some reason MSCRM is no longer reachable. Furthermore, *MSCRMExt*'s product includes pages from MSCRM. Finally, both products call upon each other's services.

MSCRMExt made the decision to create a loosely coupled product, such that their product could be used to extend other platforms in the future as well. The decision was made to maintain independence from Microsoft, even though a number of years later this has resulted in several problems. For one, due to the loose coupling *MSCRMExt* does not profit from new features that are built into MSCRM automatically, such as an extensive workflow engine, the model-view-controller pattern, and the shared repository pattern. *MSCRMExt* has a relatively small development team and therefore has trouble keeping up with the developments of the CRM platform. According to *MSCRMExt*, the MSCRM team should provide more demonstrations on new features and inform developers more frequently. Furthermore, *MSCRMExt* is not looking forward to the next version of MSCRM because of expected compatibility problems and rework.

MSCRM has made *MSCRMExt* more productive for three reasons. First, the available functionality saves *MSCRMExt* of having to develop everything themselves. Secondly, due to the fact that the look and feel is consistent throughout the platform, it is easy to reuse both Microsoft and other third-party components. Finally, there exists a large knowledge base on CRM, making it easy to find and solve problems. The documentation of the CRM platform is considered complete and of high quality.

7. ANALYSIS OF THE HYPOTHESES

Developers of platform extenders were asked how they thought the software architecture influenced the success of a platform. Most of the answers were formulated to state those things that make a platform architecture hard to adopt. Most frequently, restrictive APIs were mentioned as adoption barriers for a platform. Secondly, reuse disablers, such as the fact that it is hard to share components on the iPhone, was mentioned as a barrier to adoption. Other problems that were provided were unsecured interfaces, buggy interfaces, restrictive licensing, bad third-party components, and format incompatibility with what is standard in the industry. Interestingly enough, many of the issues presented were not of an architectural nature. Other adoption barriers that were provided are no or very expensive support, immature status of the product, insecurity whether the platform contains malware, bad documentation, a non-trustworthy supplier, the fact that the platform might be non-specific and thus too generic, and the performance of the platform might be significantly worse compared to dedicated products. A recurring issue in these discussions is that from the developer's point of view, the platform can never be open enough [10, 9].

Table 2 lists the top three reasons for each of the platform extenders for choosing the platform. Most commonly the 'available features' factor is encountered. It must also be noted that two of five platform extenders said their platform was selected because of the technology behind it. Both the RoR extenders were fed up with PHP and adopted RoR as their new development standard. For the other three platforms technology played a much smaller part and the market was considered much more important. A factor that could play a part here is that RoR is an open source and free platform, whereas the other three are closed source and commercial platforms.

Hypothesis H_1, which expects platform extenders to take architecture into account when selecting a platform, is rejected based on the findings in table 2. One of the goals of this research has been to establish whether the software architecture is considered an entry barrier for platform extenders to join a platform. When comparing two platforms, however, generally platform extenders do not really see the software architecture as impeding or encouraging the adoption of a platform.

H_2, which states that platform architectures influence the behavior of platform extenders and the development of an ecosystem, is accepted. Several quality attributes were observed that change the behavior of developers significantly, both positively and negatively. The quality attributes of platform architectures that are most frequently observed in the case studies are:

Documentation and **Learnability** - Developers complain frequently about intransparency, poor documentation, or even missing documentation in rapidly developing platform APIs. On the other hand, the platform extenders are positive about platforms for which sufficient documentation exists. Undocumented features cost developers significantly more time to work with, whereas these are typically highly martketable features, due to their novelty. Documentation platform architecture in general is weak in all four platforms, although it must be noted that the documentation in some cases specifically uses the concept of patterns to illustrate platform extension to developers.

Two quality attributes that are frequently discussed are **Backwards compatibility** and the **Rate of change** of the platform. As platforms are frequently updated, significant investments are needed from extenders to stay up to date and implement new features for the platform. Software platform developers typically fight on two fronts: on the one hand developers want their platform

to accomodate new technologies, on the other hand platforms must remain stable and reliable. The SAP platform has remained backwards compatible for almost a decade. The MSCRM platform is changing rapidly, forcing its extenders to continuously monitor and redevelop its functionality. For smaller organizations it is challenging to keep up with new platform developments while developing its own functionality. Backwards compatibility can help extenders, especially when migrating to newer platform versions require significant investment. A downside of backwards compatibility is that the platform may suffer from bloat and legacy features that continue to be supported, thereby incurring other penalties (performance, usability, etc.) on the overall platform.

Component dependency - One of the key features of a software platform are the dependency and reuse mechanisms and the openness within those mechanisms. Apple, for instance, does not allow linking of third-party libraries with iPhone Apps, whereas Ruby on Rails does. For RoR a complete dependency management system maintains dependencies, which enables platform extenders to cooperatively develop shared libraries, next to regular platform extensions. Apple's iPhone, however, does not currently implement such a dependency management system, reducing reusability of other applications by platform extenders (and power of extending parties, for that matter). These mechanisms can be taken one step further by also incuding the architectural App Store "pattern", which takes over **delivery and deployment** of extensions.

Major concerns for both the extender and the platform provider are **openness** and **accessibility**. There exists tension between the requirements of the extender and the willingness of the platform provider to publish new interfaces to existing functionalities in a commercial platform. Several reasons exist for a platform provider to restrict access to its functionality. A platform supplier will, for instance, want to guarantee its end-users a level of quality, which is reached by not allowing a platform extender access to all parts of the architecture. In the iPhone case, for instance, developers do not have direct access to the energy management of the phone. In some cases, the platform provider may even wish to restrict features to create their own competitive features, although that can also be considered cannibalistic on the ecosystem.

Finally, three quality attributes are mentioned specifically by platform extenders. First, **portability** describes the platforms that are required to install the main platform. The difference between MSCRM and its Microsoft oriented stack immediately incurs all kinds of extra license fees, whereas Ruby can be deployed on so many different configurations, it is hard for developers to pick which underlying technology fits their extensions best. The second attribute is **Standardization across the platform**, which is seen as a positive influence. In the case of MSCRM, the platform developers have done their utmost to maintain a standardized user interface across different layers in the application, thereby providing end-users with the feeling that they are always working within one product, whereas it may be possible that the end user uses several products simultaneously. One other example of standardization that is frequently praised is the use of application templates, which can easily be altered to provide extension specific features. Finally, quick and high-quality **support** for extenders is seen as having a major positive effect on an extender's experience and development speed with a platform.

For reasons of brevity, just one illustration is given of a breach of the extension mechanisms provided by the platform in the case of *MSCRMExt*. Due to *MSCRMExt*'s dependance on MSCRM, the decision was made to create a loosely coupled extension, such that future replacement by another platform remained possible. This implies that *MSCRMExt* has rebuilt large parts of the CRM platform, such as the workflow engine and a proprietary entity model, just to maintain platform independence. Furthermore, *MSCRMExt* accesses the MSCRM database directly for some data synchronization options, with its accompanying dangers of data synchronization problems, errors made in the interpretation of MSCRM's data structures, and uncaught changes. This leads to the observation that maintaining platform independence typically requires platform extenders to breach extension mechanisms.

8. CONCLUSIONS

This research looked at whether software architecture is considered an entry barrier for platform extenders to adopt a platform. When comparing two platforms, however, platform extenders generally do not see the software architecture as impeding or encouraging the adoption of a platform. It is however found that platform architectures strongly influence the behavior of platform extenders and the supporting ecosystem. Multiple examples are shown where a design decision, such as restricting access to specific libraries, affects the way in which platform extenders develop, sell, and reuse applications. Contained in this hypothesis are several lessons that can be learned by platform suppliers in regards to how the architecture of their platform can stimulate or deteriorate the ecosystem, through software platform quality attributes.

Many research challenges still exist in the area of coordination of platform extenders for platform providers: measuring adoption speed of new features by extenders, the process of making changes to architectural patterns, and the process of getting extenders to adopt a certain platform are just some of the challenges that must be tackled by software ecosystem researchers in the future.

9. REFERENCES

[1] I. Jacobson, M. Griss, P. Jonsson, Software reuse: architecture, process and organization for business success, ACM Press, NY, USA, 1997.

[2] W. B. Frakes, K. Kang, Software reuse research: Status and future, Trans. of Soft. Eng. 31 (7) (2005) 529–536.

[3] D. S. Evans, A. Hagiu, R. Schmalensee, Invisible Engines: How Software Platforms Drive Innovation and Transform Industries, The MIT Press, 2006.

[4] S. Jansen, S. Brinkkemper, M. Cusumano, Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry, Edward Elgar, 2013.

[5] M. Iansiti, R. Levien, The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability, HBS Press, 2004.

[6] J. Bosch, From software product lines to software ecosystems, in: Proc. of the 13th Int'l Conf on Sw Product Lines (SPLC), Springer LNCS, 2009.

[7] B. Iyer, C.-H. Lee, N. Venkatraman, Managing in a "small world ecosystem": Lessons from the software sector, Harvard Business Review.

[8] M. Cusumano, Technology strategy and management: The evolution of platform thinking, Commun. ACM 53 (1) (2010) 32–34.

[9] S. Jansen, S. Brinkkemper, J. Souer, L. Luinenburg, Shades of gray: Opening up a software producing organization with the open software enterprise model, Journal of Systems and Software 85 (7) (2012) 1495–1510.

[10] J. W. West, How open is open enough? melding proprietary and open source platform strategies, Research Policy 7 (32) (2003) 1259–1285.