

From Software Product Lines to Software Ecosystems

Jan Bosch

Intuit, 2500 Garcia Avenue, Mountain View, CA 94043

Jan@JanBosch.com

Abstract

Software product line companies increasingly expand their platform outside their organizational boundaries, in effect transitioning to a software ecosystem approach. In this paper, we discuss the emerging trend of software ecosystems and provide a overview of the key concepts and implications of adopting a software ecosystem approach. We define the notion of software ecosystems and introduce a taxonomy. Finally, we explore the implications of software ecosystems to the way companies build software.

1 Introduction

Software product lines can arguably be viewed as the most successful approach to intra-organizational reuse of software. Numerous companies have significantly improved the efficiency of their R&D, increased their product portfolio with up to an order of magnitude, provide consistent user experience across their product portfolio and offered levels of configurability of their products that could only be achieved through shared software components with relevant software variability. Due to these advantages, software product lines have a major business impact on the companies that manage to successfully apply the technology, i.e. from a business perspective a successful product line provides the next “S-curve” of growth for the company.

In earlier research [3], we have reported on the expanding scope of software product lines but focused on the intra-organizational context. The scope of a software product line typically evolves because it receives broader adoption within the company. However, there is no reason why a software product line would have to stop expanding at the organizational boundary. The product line architecture and shared components, referred to as the *platform* in the remainder of this paper, can also be made available to parties external to the company. Once the company decides to make its platform available outside the organizational boundary, the company transitions from a software product line to a software ecosystem.

There are, at least, two reasons why a company would be interested in moving towards a software ecosystem. First, the company may realize that the amount of functionality that needs to be developed to

satisfy the needs of their customers is far more than what can be built in a reasonable amount of time and with an R&D investment that offers an acceptable return on investment. For web services companies, as well as the software industry as a whole, the market often operates based on the “winner takes all” principle. Consequently, building a large customer base as rapidly as possible is a key strategy for long term success.

Secondly, the mass customization trend drives the need for a significant R&D investment for successful software applications. Especially on the web, e.g. web service mashups, but also in other domains, e.g. mobile devices, users demand a significant degree of customization or even compositionality that allows each user to create a potentially unique configuration that addresses her or his specific needs and desires. Extending the product (which includes the platform) with externally developed components or applications provide an effective mechanism for facilitating mass customization.

The aforementioned developments are some of the driving forces behind the emergence of software ecosystems. For instance, companies that have initially built success with their web application are compelled to platformize their application and open it up for contributions by third party developers. These developers can provide functionality that addresses the needs of user segments that the company providing the platform would be unable to develop by itself.

Although companies have different reasons for adopting a software ecosystem approach, one can identify a number of convincing arguments explaining the current trend:

- Increase value of the core offering to existing users
- Increase attractiveness for new users
- Increase “stickiness” of the application platform, i.e. it is harder to change the application platform
- Accelerate innovation through open innovation in the ecosystem
 - Collaborate with partners in the ecosystems to share cost of innovation
 - Platformize functionality developed by partners in the ecosystem (once success has been proven)
- Decrease TCO for commoditizing functionality by sharing the maintenance with ecosystem partners

This paper discusses the emerging trend of software ecosystems in more detail and provides an overview of the key concepts and implications of adopting a software ecosystem approach. The contribution of this paper is threefold. First, it identifies and illustrates the increasing importance of software ecosystems. Second, it defines the notion of software ecosystems and provides a taxonomy. Third, it discusses the implications of software ecosystems to the way companies build software.

The research reported in this paper is based on action research (participant observer) by the author as well as an extensive study of software ecosystems performed by the author. Several examples including existing products and companies are provided, but no detailed references are provided. All examples are well known in the industry and can easily be found using an internet search engine. The work reported on in this paper is based on publically available information.

The remainder of the paper is organized as follows. The next section defines the notion of software ecosystems in more detail and provides a taxonomy of software ecosystems. Subsequently, the next section discusses the implications for a company transitioning from a software product line to a software ecosystem. Then, section 4, discusses the consequences on the approach and process that software companies are required to implement when adopting a software ecosystem approach. Finally, the paper is concluded in section 5.

2 Software Ecosystems Taxonomy

The notion of ecosystems originates from ecology. One definition in Wikipedia defines an ecosystem as a natural unit consisting of all plants, animals and micro-organisms (biotic factors) in an area functioning together with all of the non-living physical (abiotic) factors of the environment [4].

Although the above is an excellent definition, it is less suitable for the discussion in this paper and therefore we start from the notion of human ecosystems. A human ecosystem consists of actors, the connections between the actors, the activities by these actors and the transactions along these connections concerning physical or non-physical factors. For the discussion in this paper, we further distinguish between commercial and social ecosystems. In a commercial ecosystem the actors are businesses, suppliers and customers, the factors are goods and services and the transactions include financial transactions, but also information and knowledge sharing, inquiries, pre- and post-sales contacts, etc. Social ecosystems consist of users, their

social connections and the exchanges of various forms of information.

A software ecosystem consists of the set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions. Of course, a software ecosystem is also an ecosystem, specifically a commercial ecosystem, and hence the goods and services are the software solutions and software services that enable, provide support for or automate activities and transactions.

For example, an activity for a small business is time tracking of employees, needed for payroll data. An example transaction is a payment from one business to another business. These activities and transactions can be supported, i.e. the task is simplified and made less effort intensive, or automated, i.e. the task is performed by software solutions without any human interaction. A third category is the enablement of tasks that either are impossible or prohibitively expensive without (networked) software solutions. The emergence of online social networks is an illustrative example of the enablement of new possibilities.

Although software ecosystems receive significant attention in the Web 2.0 context, this category of ecosystems has been around for decades. Especially illustrative is the early 1990s when different companies jockeyed for dominance of desktop operating systems, gaining the support of the most influential and largest number of 3rd party developers was recognized as crucial and companies like IBM and Microsoft explicitly managed to this dimension.

end-user programming	MS Excel, Mathematica, VHDL	Yahoo! Pipes, Microsoft PopFly, Google's mashup editor	<i>none so far</i>
application	MS Office	SalesForce, eBay, Amazon, Ning	<i>none so far</i>
operating system	MS Windows, Linux, Apple OS X	Google AppEngine, Yahoo developer, Coghead, Bungee Labs	Nokia S60, Palm, Android, iPhone
category platform	desktop	web	mobile

Figure 1. Software Ecosystem Taxonomy

The above example illustrates an operating system centric operating system. However, software ecosystems exist in multiple areas and in figure 1 a taxonomy is presented that aims to organize these ecosystems in a two dimensional space. The first

dimension is best described as the abstraction level at which the software ecosystem exists, defined at three levels, i.e. operating system, application and end-user programming. The second dimension captures the evolution of the computing industry in terms of the dominant computing platform, i.e. desktop, web and mobile. Although one might argue that before the desktop, mainframes and mini-computers existed and that, in the era of ubiquitous computing, one can define other platforms besides mobile, the lion's share of software development takes place in the scope defined in this taxonomy. Below, we discuss each category in more detail.

Operating System-centric Software Ecosystems

This category is the first one where software ecosystems were explicitly identified and managed. Illustrative examples of this category are Windows, Linux and Apple OS X. Although Windows currently is the dominating operating system, in the early 1990s, it was competing for market share with IBM's OS/2 and Mac OS. During more recent years, Linux and Apple OS X have gained market share for especially the server and client desktop, respectively.

On the web, several platforms are aspiring to be the operating system of the web. On the server side, platforms that are available include Google AppEngine, Yahoo developer, Coghead and Bungee Lab. All platforms assume the use of a browser at the client side, potentially with one or more plug-ins.

For mobile devices, several platforms exist and have had waves of success, e.g. Palm OS during the 1990s. However, over the last years, the competition has intensified and the platforms currently competing for the dominant position include Symbian/Nokia Series 60, iPhone and Google's Android.

Characteristics

This category has the following characteristics:

- Operating system based software ecosystems are domain independent and assume that third party developers build applications that offer value for customers.
- In the case of desktop and mobile, the operating system is installed for every device that wants to run the applications developed for the ecosystem. Hence, the success of the operating system ecosystem is heavily influenced by the sale of devices with its operating system implemented.
- Operating systems typically optimize for the deployment of stand-alone applications and offer little support for cross-application integration and composition.

- Operating system providers offer development tools for the ecosystems to simplify the adoption of the operation system by developers.

Success Factors

The following success factors can be identified for this category:

- The success of the operating system ecosystem is defined by the applications built on top of it. Hence, a key success factor is minimizing the effort required by developers to build applications on top of the operating system.
- Although the operating system provides generic functionality, it constantly needs to extend its set of features in order to maintain attractiveness for developers. The ability of an operating system to incorporate commoditizing functionality early without alienating the existing developers is an important factor for success or failure.
- Finally, perhaps the most important success factor is the number of customers that use the operating system and that are accessible to developers for monetization.

Challenges

This category needs to address the following challenges:

- The complexity of different hardware configurations easily becomes a major source of inefficiency. For desktop and mobile, the various types of devices, because of the differences in physical configuration, may easily break compatibility of applications. Especially in the mobile domain where rapid evolution of hardware capabilities takes place, the balance between exploiting the latest and greatest features of an individual device type and backward compatibility with older device types is very difficult to maintain.
- After the enormous financial success of the Microsoft Windows platform, there is significant weariness in the industry to allow a single player become the dominant player on the web or in the mobile domain. Hence, the struggle for dominance may require sacrificing the direct financial benefits that accompany a dominant platform, e.g. the decision by Nokia to open-source the Symbian/Series 60 platform.

Application-centric Software Ecosystems

Software ecosystems organized around an application can, in a way, be viewed as opposite to the operating system-centric ecosystems. This category is domain-specific and often starts from an application

that achieves success in the market place without the support of an ecosystem around it. This first wave of success creates a, preferably large, set of customers and a healthy financial foundation for any company succeeding in the domain.

The success of the application in the market place tends to generate a large amount of specific requests that the company is not able to satisfy due to its limited R&D resources and because the business model typically does not allow for features that are used by a subset of the customers. The response from companies typically is to provide APIs so that customers can, initially, hire software engineers to extend the application with customer-specific functionality. Once the company starts to open up the application through the provisioning of APIs, the application turns into a domain-specific platform that 3rd party developers can extend to build extensions or bridges to other applications that increase the value to a subset of the customers. Assuming the company transitions successfully from an application to a platform approach, the creation of the software ecosystems provides the foundation for a second period of aggressive growth.

Examples of successful application-centric software ecosystems on the desktop such as Microsoft's Office suite and Autodesk's AutoCad product have been around for years and significant value has been created for the platform company and for the 3rd party developers building solutions on top of the platform. However, the new trend that emerged during recent years is that especially on the web, in the Web 2.0 Software-as-a-Service area, many companies are explicitly driving a software ecosystem strategy as soon as they have engaged a sufficiently large group of customers. Examples include Salesforce, eBay and Facebook. These companies first built a successful application with significant customer adoption and subsequently opened up their application for third party developers. It is interesting to note that no application-centric ecosystem has been established on the mobile computing platform.

For any company that has managed its product development in house evolving its product to become a platform for a software ecosystem brings a host of changes to the business strategy, architecture, development processes and R&D organization. In the experience of the author, although organizations realize this at a rational level, in practice the required changes often are implemented only half-heartedly and the engineering staff easily falls back into old patterns and approaches.

Characteristics

This category has the following characteristics:

- As discussed above, the application-centric platform typically starts from a successful online application.
- The platform provider frequently provides hosting or another technique to make the experience between the platform and the 3rd party application as seamless as possible for the customer.
- Third party application developers extend domain-specific functionality provided by the platform. This is different from operating-system centric ecosystems where developers build their own applications.
- To achieve seamless integration and added value for the customer, deep integration between the platform and extensions is facilitated for data, workflow and user experience.

Success factors

The following success factors can be identified for this category:

- The foremost success factor for any application-centric software ecosystem is a large set of customers or the promise of those customers that have real reason to extend the platform with additional functionality. Although the "coolness" factor certainly plays a role, third party developers are driven by their business case, i.e. a pool of potential customers.
- Assuming the first factor is met, the platform company should aim to simplify contributing by third party developers through allowing the use of generic, popular development environments, stable and expressive interfaces and, in the case of web applications, easy deployment and integration with the platform.
- Although most application platform companies focus on providing access to data, providing solutions to extend data models and workflows as well as integrate in the same user experience framework is important to achieve a seamless integration from a customer's perspective.
- Finally, third party developers need paying customers and the platform company needs to take a role in providing a viable channel where 3rd party contributions are exposed to customers.

Challenges

This category needs to address the following challenges:

- The primary challenge that application platform companies struggle with is the conflict between the product strategy and the platform strategy. The

company has enjoyed its first major success focusing the entire company on a product strategy. It now needs to transition towards a platform strategy and there are profound changes between both strategies. From a product strategy perspective, a platform strategy does not drive new value for customers, as it mostly enables third party developers to build this new value. Secondly, the freedom of changing APIs, user interfaces, data models, etc. is significantly constrained by the platform strategy and especially initially platform companies may easily cause binary breaks to happen as the relative priority of the platform strategy is not yet sufficiently established.

- At least anecdotal evidence from the blogosphere and news articles seems to suggest that establishing a viable business model for third party developers proves an elusive goal that is difficult to achieve for most software ecosystems. For application-centric ecosystems, there is the additional challenge that the application platform has to be sufficiently useful to customers to acquire it on its own merit. Because of that, the number of customers that actively looks to extend the platform with additional functionality may be limited.

End-User Programming Software Ecosystems

The third category of software ecosystems is smaller and of less importance compared to the former two. On the other hand, it has for a long time been the holy grail of software platforms, e.g. [4], i.e. providing a configuration and composition environment that is so intuitive (in other words modelled in terms of the concepts of the end user) that the end user can create the required applications herself.

Most end-user programming ecosystems take the form of a domain specific programming language (DSL), either graphical or textual. In practice, the DSL captures the points of variation and composition in the underlying application domain-specific platform and exposes these through a tool that facilitates configuration and composition. Because of the amount of work required to create a DSL and all the tooling around it, the scope of the DSL and hence the space within which end-users can create applications tends to be limited. One perspective is that this category of software ecosystems is more narrow and focused than application-centric software ecosystems (that in themselves already are more focused than operating system-centric ecosystems). However, it often is surprising to see how creative committed end users can

create applications way beyond the intended scope of the DSL.

Illustrative examples of end-user programming ecosystems include Microsoft Excel, Lego Mindstorms and stream-based mashup creation tools such as Yahoo! Pipes and Microsoft PopFly. Again, one has to observe that no successful end-user programming software ecosystems exist for the mobile domain.

Characteristics

The characteristics of this category are as follows:

- The category is explicitly focused on application developers that have good domain understanding, but no computer science or engineering degree. Hence, the basics of computing and programming cannot be assumed to be understood by this group.
- Although exceptions exist, especially the end user programming environments for the web tend to be based on the pipes-and-filters architecture, assuming a stream of data that is processed and presented.
- From the perspective of a software engineer, end-user programming often is concerned with creative composition of pre-created building blocks, rather than the creation of fundamentally new functionality.

Success factors

The following success factors can be identified for this category:

- The primary success factor is the amount of value for him or herself that an end user generates through the creation of a unique, customer-specific solution. If the added value of the unique solution is not perceived as sufficient, the ecosystem will have difficulty in becoming successful.
- As the actual creation of applications will only be done by a, most likely small, subset of the total set of customers, providing effective ways of sharing applications in ways that allows other customers to adopt, adjust and use these solutions is critical. As this category of ecosystem often is low in financial value, the motivation for end-users to engage in application creation and sharing is prestige and status in their community.

Challenges

This category needs to address the following challenges:

- The biggest challenge, by far, is to model the application domain in terms that the end user will understand with minimal instruction and to make application creation intuitive and requiring

minimal understanding by the end-user. Many domains are simply too complicated to create a domain specific language and associated configuration and composition environment that covers the needs of a sufficiently large number of customers.

- The domain in which the end user ecosystem is created needs to be sufficiently stable to allow for low maintenance effort for the tooling infrastructure that is created for the domain.

3 Transitioning to a software ecosystem

Software ecosystems are a logical next step for a company that has a successful platform and intra-organizational software product line. The best approach towards the introduction of a software ecosystem depends on the type of product line.

In this section, we first discuss the arguments for selecting the type of software ecosystem that a product line company adopts. Subsequently, we discuss the approaches available to opening up the platform.

Selecting the software ecosystem type

For most companies, the most likely scenario will be to introduce an application-centric software ecosystem. On the one hand, introducing a new operating system and building an ecosystem around it is an extremely difficult challenge. The primary reason for this is that one needs to convince developers to build compelling applications on the operating system in order to attract customers. On the other hand, without customers willing to acquire applications, few developers are willing to invest in application development. Getting this flywheel going is difficult for a variety of reasons and there is a long history of failed attempts.

The introduction of an end-user programming ecosystem is similar in its challenge in that there already has to exist a large set of users of the platform, which implies that the platform provides intrinsic, domain specific value to end users. Only when the number of users is sufficiently large and the number of unique configurations required to satisfy the needs of small segments of users is large as well is there a case to be made for end-user programming solutions.

In addition, there is insufficient value for the platform provider to optimize the programming infrastructure for external parties to the extent that end-users could build or compose their own applications. Hence, the approach potentially introduced gradually is to open the platform to external developers.

This leaves the application-centric software ecosystem as the most viable option for most companies employing a software product line. Since there is a range of products derived from the product line, there is a customer base that is potentially attractive to 3rd party developers. In addition, the products provide value to their users, so even if the user never acquires or uses an application to extend the original product, there still is value for the customer.

Opening up the platform

Once the company decides to transform the product line into a platform for external developers, again several decisions need to be made. The first, possibly biggest, decision is deciding to what extent the company takes a directed versus undirected approach to partner and application selection.

The directed approach assumes that the platform company has identified specific areas of functionality that it is unable or unwilling to develop itself, but would like to offer to its customers. In this case, the company selects partners able to provide solutions for those selected areas and negotiates an agreement with that partner. The agreement typically includes a form of revenue sharing and the partner company gains deep access to the platform and the products developed internally on top of that platform. Note that an agreement in which the partner is reimbursed for its R&D expenses and then transfers the intellectual property rights to the platform company is traditional outsourced development and is different from the situation discussed here. The directed approach allows the platform company to rapidly, with low risk and against a low initial R&D expense expand the scope of functionality offered to its customers, at the expense of revenue shared with its partners.

The undirected approach can be viewed as being at the other end of the spectrum: the platform company offers the platform as a basis for application development by external developers without constraining neither who gets to develop applications for the platform nor what gets built. The external developers can build solutions that compete with each other as well as with the platform company itself. Although it may seem counter-intuitive to have the external development partners compete directly with the platform company, the underlying assumption is that competition is the best mechanism to deliver the best solutions to the users as it is them, as customers, which make the final decision on which selection of solutions to use in their daily practice.

The directed and undirected approach are two extremes and ignore an important aspect: it is possible, and even preferable, to have multiple tiers of

developers. For a typical product line transforming to a software ecosystem, there would typically be four tiers of developers:

- **Internal developers:** Although the company has decided to open up the platform to external developers, the platform still serves internal developers in the product teams as well. As the product and platform teams are part of the same organization, the level of trust and ease of communication is such that internal developers can be offered deep access and insight into the platform.
- **Strategic developers:** The strategic developers are those companies that have long-term relationship and that have explicitly been selected for partnerships. These developers, typically under the directed model, extend strategic areas of functionality. Due the nature of the relation, strategic developers have deep access to the platform, understand the long term roadmap for the platform and overall product line and their software is considered, with some validation, as trusted and hence has read and write access to, often private, data stored by users.
- **Undirected developers:** These developers build their own solutions and aim to market and sell these to the users of the basic platform and products to extend the functionality provided by the platform company and strategic developers. As it often is difficult to guarantee the absence of accidental or even intentionally malicious code in the solutions provided by undirected developers, these developers have constrained access to the platform and product functionality. Although undirected developers are unpredictable and sometimes difficult to manage, it typically is this group that is able to use the platform in totally unforeseen ways and hence provide a significant innovation impulse in the overall ecosystem.
- **Independent solution vendors:** The final category is not necessarily unique to a software ecosystem, but relevant in this case as well. The independent solution providers (ISVs) offer customer specific integrations of one or more products provided by the product line company with other IT solutions to build a solution that optimally solves the specific needs of a unique customer. Maintaining a good relationship with the ISVs is important for the platform provider as an important source of future requirements is provided by the customer specific solutions that are common between multiple customers. Commonality between customer needs that is not

captured by the platform and products on top of the platform drives the roadmap for these.

Building the developer relationship

One of the challenges for a company transitioning from a product line to a software ecosystem is to build and evolve the relationship with the external developers. As the company has achieved its success by selling products with functionality that customers care deeply about, there is a natural tension with allowing external developers to become involved in that process and, to some extent, even taking over part of the customer relationship.

Secondly, the platform company needs to evolve its platform and incorporate new functionality. One main source of new functionality is found in the solutions developed by external developers. Unless this process is managed carefully by the platform company, the incorporation of functionality into the platform that earlier was provided by external developers will be contentious and cause a negative reputation for the platform company. On the other hand, the platform company cannot avoid the evolution of the platform without risking the platform to become irrelevant over time.

The best approach is often to publicly release a long term platform roadmap that indicates the intentions of the platform company and allows external developers to move towards more differentiating functionality and abandoning the area where the platform company plans to move into.

4 Implications for software engineering

The transition from a product line centric approach to a software ecosystem approach can be considered as comparable to the shift from a product-centric to a product line centric company. The implications affect virtually every function in the company, as we indicated in the previous section.

The way the company engineers its software, however, is affected even more than the other functions of the organization. The implications can be categorized in three main areas, i.e. coordination mechanisms, engineering agility and product composition. In the sections below, each area is discussed in more detail.

Coordination mechanisms

The first main change between internal development and the software ecosystem approach is that external development teams cannot be subjected to standardized process models, tools and ways of

working. This means that traditional process maturity approaches, such as CMMi [1], become much more difficult to apply in this approach.

On the other hand, the number of parties involved and the complexity of the relationship between these parties is complicating the overall interaction model to an extent that a traditional, centralized model will be difficult, if not impossible to manage. The complexity and consequent coordination overhead is such that the competitiveness of the approach comes in question.

This concern does not just address the software development processes, but also the end-to-end processes including the management of requirements and roadmapping initiatives. The adoption of a software ecosystem causes processes optimized for intra-organizational purposes to no longer work, or at least to be much less effective, and hence alternatives need to be found.

The high-level solution to this challenge is to move from a centralized to a decentralized approach. Traditional software development tends to rely heavily on centralized mechanisms, e.g. around requirements management, architecture evolution, integration, quality assurance and SCM. At these stages, all assets involved in a larger system as well as all teams responsible for these assets need to synchronize and coordinate. The logical solution, then, obviously is to decentralize these activities and to move to a much more compositional approach to software development. In earlier work [6], we have promoted the compositional approach in the context of large-scale software product lines, but in the case of software ecosystems, a compositional approach is no longer an option but rather a necessity.

The implications of a decentralized, composition-oriented approach to software development, however, are profound. The first major change is that the coordination that is still needed across the system is achieved through the software architecture, rather than the development processes. The overall decomposition of the system, the underlying design decisions, interfaces defined for the components and the non-functional requirements provide the context in which individual teams can develop their solutions without having to coordinate with other teams or a central function. The architecture provides a formalization of the rules of interoperability and hence teams can, to a large extent, operate independently.

The second change is that centralized requirements management and roadmapping are replaced with bottom-up, team driven roadmaps and requirement specifications. This situation certainly exists between the platform company and its external developers, but some companies apply the same approach internally as

well. This means that each component team announces its own roadmap and the requirements that it will release at the end of the next iteration cycle. New functionality may affect the provided and required interfaces between components, requiring the component team to reach out to other teams for local discussions on interfaces.

The final aspect that we discuss in this section is the changing role of software configuration management and quality assurance. Rather than focusing on the overall system functionality, these functions address compositionality and backward compatibility requirements instead. This allows decentralized quality assurance and minimizes the amount of effort that needs to be spent centrally. A second major advantage of focusing on backward compatibility is that it significantly simplifies the management of versioning: when a new version is validated, all older versions can be retired as backward compatibility is guaranteed. Of course, to achieve the latter, the interfaces between components need to be narrow and allow for minimal interdependency, e.g. using web service style approaches.

Engineering agility

The second main change in the approach to software development is that the paradox between agility of the overall software development capability and the challenge of frequent platform releases needs to be managed carefully.

As we reported in [3], one of the challenges in software product lines with broadening scope, including those transforming into a software ecosystem, is the decreasing frequency of platform releases. The number and complexity of dependencies between different components and their associated teams tends to be such that a centralized quality assurance effort will require an increasing amount of effort.

In the previous section, we discussed the importance of decentralization and several examples exist where small teams can operate very effectively even in the context of large and complicated systems.

For platforms where there is no business case for frequent iterations of all layers of the architecture, one approach is to allow for different release frequencies for different layers of the stack.

Product composition

The third and final main area of impact on the software engineering practices for an organization transitioning from a software product line to a software

ecosystem is the change in ownership of the product composition.

A software ecosystem consists of a platform, products built on top of that platform and applications built on top of the platform that extend the products with functionality developed by external developers. However, the immediate consequence of this approach is that the party composing the functionality of the overall solution is no longer the product line company, but instead the customer. The customer composes the solution that best suits his or her needs and assumes the resulting selection works seamlessly without requiring any additional work.

Although this seems trivial in theory, in practice there are many configurations that cannot be tested and hence the architecture and overall guidelines need to guarantee compositionality and a best-effort composition of selected functionality.

The second challenge in this space is maintaining a compelling user interaction experience. As users are composing their own solution consisting of elements built by different parties, the overall user experience can easily suffer, resulting in a less attractive offering for the customer. The platform company has to provide a basic user experience framework that addresses this concern. Although no solution exists that solves this challenge completely, the platform company can provide significant support to minimize the concerns.

5 Discussion

Software product lines can arguably be viewed as the most successful approach to intra-organizational reuse of software. Companies successfully adopting software product lines have managed to use their product line to provide the next wave of growth for the company.

In earlier research [3], we have reported on the expanding scope of software product lines but focused on the intra-organizational context. However, there is no reason why a software product line would have to stop expanding at the organizational boundary. Once the company decides to make its platform available outside the organizational boundary, the company transitions from a software product line to a software ecosystem.

In this paper, we discussed the implications of a software product line company transitioning to a software ecosystem approach. We first discussed the emerging trend of software ecosystems in more detail and provided a taxonomy organizing software ecosystems in a two dimensional space. The first dimension is best described as the abstraction level at which the software ecosystem exists, defined at three

levels, i.e. operating system, application and end-user programming. The second dimension captures the evolution of the computing industry in terms of the dominant computing platform, i.e. desktop, web and mobile.

Subsequently, we analysed the actions required to transition to a software ecosystem approach and highlighted the importance of selecting the appropriate software ecosystem type, the approach to opening up the platform and the development of the relationship with developers.

Finally, we discussed the implications for the approach to software engineering that transitioning companies need to consider. These include the evolution of coordination mechanisms, maintaining engineering agility despite significant counter forces and addressing the issues surrounding customer-driven product or solution composition.

Although software ecosystems are discussed frequently in blogs and popular media focusing on web 2.0, little research literature exists that addresses the domain of software ecosystems. An early publication is [6] although several aspects of discussion in this paper are not addressed there. With the increasing adoption of software ecosystems, it is important that the area receives more attention from the research community. As software product line companies are likely candidates for making the transition and there are significant parallels between product lines and software ecosystems, this research community would be particularly suited for addressing this topic.

The purpose and contribution of this paper is to increase the visibility of the software ecosystems trend and provide an initial theoretical framework for reasoning about different classes of ecosystems. Secondly, it discusses the implications of software ecosystems to the way companies build software.

In future work, we intend further develop the theoretical context and to study more industrial cases of companies transitioning to software product lines.

6 References

- [1] D. Ahern, A. Clouse and R. Turner, CMMI Distilled: A Practical Introduction to Integrated Process Improvement, ISBN: 0-201-73500-8, 2001, Addison Wesley, 2001.
- [2] J. Bosch, "Software Product Families in Nokia", Software Product Lines Conference (SPLC 2005), Springer-Verlag, 2005, pp. 2-6.
- [3] J. Bosch, The Challenges of Broadening the Scope of Software Product Families, Communications of the ACM, Volume 49, Issue 12, pp. 41 – 44, December 2006.

- [4] <http://www.smart-generators.org/DSLWC>
- [5] <http://en.wikipedia.org/wiki/Ecosystems>
- [6] Messerschmitt, D. G. & C. Szyperski, Software Ecosystem: Understanding an Indispensable Technology and Industry, MIT press, 2003.
- [7] Christian Prehofer, Jilles van Gurp and Jan Bosch, Compositionality in Software Product Lines, in Emerging Methods, Technologies and Process Management in Software Engineering, Andrea De Lucia, Filomena Ferrucci, Genny Tortora and Maurizio Tucci (eds.), pp. 21-42, Wiley, February 2008.
- [8] Software Product Line Hall of Fame http://www.sei.cmu.edu/productlines/plp_hof.html