A Sense of Community: A Research Agenda for Software Ecosystems

Slinger Jansen
Utrecht University
s.jansen@cs.uu.nl

Anthony Finkelstein
University College London
a.finkelstein@cs.ucl.ac.uk

Sjaak Brinkkemper
Utrecht University
s.brinkkemper@cs.uu.nl

Abstract

Software vendors lack the perspective to develop software within a software ecosystem. The inability to function in a software ecosystem has already led to the demise of many software vendors, leading to loss of competition, intellectual property, and eventually jobs in the software industry. In this paper we present a research agenda on software ecosystems to study both the technical and the business aspects of software engineering in vibrant ecosystems. The results of such research enable software vendors to develop software that is adaptable to new business models and new markets, and to make strategic choices that help a software vendor to thrive in a software ecosystem.

1. Software Ecosystem Perspectives

Software vendors no longer function as independent units, where all customers are end-users, where there are no suppliers, and where all software is built in-house. Instead, software vendors have become networked, i.e., software vendors are depending on service and software suppliers, value-added-resellers, pro-active customers who build and share customizations, and many others. Software vendors now have to consider their strategic role in the software ecosystem (SECO) to survive.

SECOs introduce many new research challenges on both a technical and a business level. In a traditionally closed market, software vendors are now facing the challenge of opening up their product interfaces, their knowledge bases, and in some cases even their software. Software vendors must decide how open their products and interfaces are, new business models need to be developed, and new standards for component and service reuse are required. These challenges have been identified but have hardly been picked up by the research community [3]. This research agenda presents both SECO research challenges and a number of research examples on SECOs. Many of the research challenges have been inspired by the book of Messerschmitt and Szyperski on SECOs [6] and our own research [3, 1, 4].

Software vendors have to focus on three different perspectives: the *software ecosystem level*, the *software supply network level*, and the *software vendor level*. On the *software ecosystem level* strategic choices must be made on how a software vendor behaves in a SECO to maximize profitability. SECO orchestrators, for instance, have control over the SECO and can develop strategies to keep a SECO vibrant and profitable for other organizations in the SECO. On the *software supply network level* software vendors determine strategies in regards to their immediate buyers and suppliers, by for instance organizing regular meetings with plug-in builders, by developing customer and reseller portals, etc. On the *software vendor level*, vendors establish the effects of the SECO on their product and service portfolio, knowledge management, and relationship management. As an example, guidelines must be developed regarding make-or-buy decisions and reuse, the software vendor must continuously decide on exploitation of its complete product and service portfolio, and the software vendor must determine how it uses knowledge from the SECO internally.

The three perspectives are modeled in figure 1. The model is created using the software supply network modeling technique developed by Brinkkemper et al. [1] and models a software vendor (ISV) in the middle. The software vendor is supplied with components from an outsourcer and another software vendor. Furthermore, the vendor supplies a product (P.1) and a service (S.2) to its customers. Some competition exists, in the form of an ISV who delivers to other customers and is supplied by a software vendor. The *software vendor level* encompasses all products and services supplied by the vendor and the vendor itself. The *software supply network level* considers all buyers and suppliers who are in direct contact with the software vendor. Finally, the *software ecosystem level* encompasses all related software organizations in the SECO.

2. Software Ecosystems Level

We define a software ecosystem as a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships

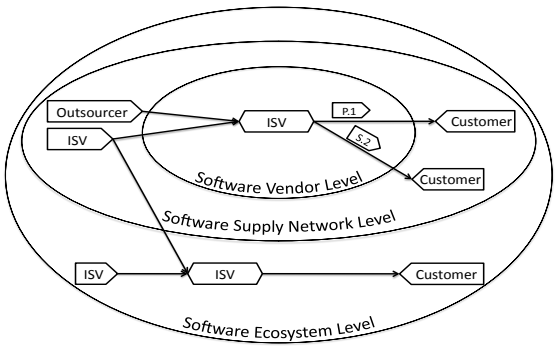

Figure 1. Software Ecosystem Perspectives

among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts. Some examples of SECOs are the MySQL/PHP SECO, the Microsoft SECO, and the iPhone SECO. These examples can be used to establish typical characteristics of SECOs; SECOs can be contained in other SECOs, such as the Microsoft CRM SECO that is contained in the complete Microsoft SECO. Also, one might refer to the iPhone SECO with its AppStore as a closed SECO, whereas the MySQL/PHP SECO is open, since organizations have access to its source code and related knowledge bases.

SECO Challenge 1 - Characterisation and modeling of SECOs. To model SECOs formal characteristics of these SECOs need to be established. Some example characteristics are size, livelihood, the presence of standards or standards organizations, and the different roles that software vendors can take within these SECOs. As an example, Microsoft and Apple are SECO leaders and orchestrators, whereas game developers for the iPhone and its AppStore are SECO followers. The modeling challenge lies in the fact that no modeling formalisms currently exist for SECOs [1].

SECO Challenge 2 - Developing policies and strategies within SECOs for SECO orchestration. These policies determine activities, guidelines, standards, and actions that can be taken to influence the SECO. Some examples of policies are standardization efforts, certification, and common delivery channels (in the case of the AppStore example the iPhone and iPod-Touch are the sole delivery channels). Strategies include lowering prices for active SECO participants, introducing developer conferences and workshops, introducing democracy in SECOs, and developing common channels for communication, marketing and development.

SECO Challenge 3 - Determining a strategy to thrive and make profit in an SSN. Questions that arise are "Should we go open source?", "Should we deliver our products and services to small or large numbers of customers?", "Are we better at building software or at exploiting software?" The true challenge lies in the design of the business model of the software vendor. To establish a business model, software vendors require modeling techniques to model the software functionality portfolio, the software deployment context, the software supply network, and the financial model [5]. These models assist software vendors in establishing their different sales and distribution channels, their main competitors, and their potential partners.

Example: Modeling Ecosystems - Before steering mechanisms can be developed for a software vendor in a SECO, a modeling technique must be devised for SECOs. In this research a start is made by modeling a small SECO and its influencing factors. The model is presented to several software vendors who are active in the SECO, to validate the usefulness of the SECO models. A second step in this research will be the development of a vibrance indicator for an SECO, to enable software developers to choose whether to become active within the SECO.

Example: Create your own Ecosystem - A software company in the Netherlands that supplies traditional enterprise resource planning products has recently developed a design language to generate new custom solutions quickly and easily. The design language enables customers and partners of the company to develop new partial solutions and sell them in the component shop of the software company. The software vendor is changing from a SECO follower to an SECO leader. In this research we are determining what the success factors are for such a component store and what other activities are required to get participants in the SECO to start building with the design language.

3. Software Supply Network Level

A Software Supply Network (SSN) is a series of linked software, hardware, and service organizations cooperating to satisfy market demands [5]. At the software supply network level software vendors must consider how to deal with first-tier buyers and suppliers and in many cases even their buyers and suppliers (second-tier to n-tier). The challenges consist of strategically deciding who will belong to the groups of buyers and suppliers, governance of relationships with buyers and suppliers, and cross-organizational quality assurance.

SSN Challenge 1 - Establishing relationships in a SSN. Methods for partner contracting and relationship identification are required to further assist software vendors in establishing and developing their own SSNs. Furthermore, software vendors need to keep relationships with (potential) buyers and suppliers alive. One way to do this is by using different web portals for end-users, suppliers, and (value-adding) resellers. Other ways of doing this are by establish-

ing close relationships with partners by organizing meetings, workshops, and events surrounding the technology and business opportunities. Furthermore, policies must be established on how software vendors present themselves to potential buyers and suppliers in forums, at events, through marketing channels, etc. These challenges affect the whole software vendor's organization, from marketing to support and from management to development.

SSN Challenge 2 - Release heartbeat and release timing. When considering dependencies between components developers want to have the newest software as soon as possible, so they can reuse the newest features. On the other hand end-users demand stable systems with a clear return on investment for upgrading. Research is required on factors that determine the optimal functionality release moment for software vendors and all participants in a SSN.

SSN Challenge 3 - Managing quality in the SSN. An interesting development is that software vendors require their plug-in developers to maintain certain quality levels to deserve an approved status [7]. After all, customer approval of a software product or service are dependent on the experiences with both the plug-ins and the main product or service. Several challenges are faced when establishing such guidelines, since quality standards can easily become too rigid for plug-in developers. The freedom of a plug-in developer is of course dependent on how strongly the developer is locked-in with the SECO leader.

Example: Breaking Open the Business Model - A traditional Dutch software company has recently released a new version of their software. Their product, a building design application, is currently monolithic and delivered on a CD-ROM to customers. The software company is presently exploring options for turning its product or parts of its product into services and components, to be sold separately. If successful, the software vendor can function as an example in its relatively traditional market.

Example: Outsourcing Governance - Types of development outsourcing relationships are established by the specific needs of a company. These needs depend on clarity of the concepts and requirements that must be developed, recurrence of development tasks, and size and strategic strength of buyers. In this research we attempt to develop a method for outsourcing governance, such that an organization wishing to outsource a development task can use the method to establish and govern relationships.

Example: Leveraging the Ecosystem - Recently a Dutch software vendor has opened up its business model, in that the software vendor has enabled partners to build components and plug-ins for their product. A study that is going to be conducted is how the knowledge infrastructure is designed to share knowledge with those partners and how interfaces are opened up to these partners. Some objects of the study are the knowledge infrastructure, the vibrance of the SECO, the interfaces and openness of the software, and the plug-in builders.

4. Software Vendor Level

A software vendor is an organizational entity that designs, builds, and releases software functionality within a SECO. The aim of a software vendor is to maximize profits by selling software and possibly related services. Software vendors build software functionality. Software functionality is any collection of functions that represent business value for a software end-user. Software developers are increasingly separating functionality from the method in which the functionality is distributed, i.e., software functionality is built to be run independently, to be embedded as a component, and to be published as a service. This enables software vendors to distribute their functionality through services, products, components, and libraries to different customers. The separation of functionality and distribution method enables software developers to avoid the stigma of being either a product software vendor or a service provider [2]. Furthermore, the separation allows vendors to quickly change their main distribution methods, as determined by their strategic planning in regards to the SECO. Software vendors are faced with the challenges of portfolio and product line planning, knowledge management, architecting for extensibility, and integration of development support systems across different products and services.

Software Vendor Challenge 1 - Portfolio and Product Line planning. Software vendors need to decide in which configuration new functionality will be released (release and portfolio planning) and how much this functionality is worth. Software vendors must develop a strategy for how independent product development departments develop and release their software in unison. Furthermore, software vendors must constantly reconsider their standards in regards to software reuse and make-or-buy decisions.

Software Vendor Challenge 2 - Knowledge Management. A vendor needs to decide how much information shall be shared with other participants in the SSN, how open interfaces will be, what types of software feedback will be shared within the SSN, and how software will be made available to new participants. The vendor also needs to determine how much knowledge and software from the SECO will be reused, such as open source software functionality. Software vendors must establish clear guidelines for developers on the types of software and licenses that can be reused. These guidelines can counter the pragmatic nature of software reuse [4]. Another area of knowledge management is the use of software operation knowledge, which is using software performance, usage and feedback data to support the software development and maintenance processes [9]. Current challenges in this area are mining soft-

ware feedback data, visualizing information derived from feedback data, and the determination of processes for which feedback information can be used. Another interesting challenge in regards to knowledge management is the sharing of bug reports: should a software vendor inform all parties in the SECO of a bug?

Software Vendor Challenge 3 - Architecting for extensibility, portability, and variability. Architects are not supported, as in other industries, by industry standards that enable the software architect to use uniform interfaces. Many challenges exist in this area: how flexible must the architecture of software be? Can domain specific software design languages and standards be created? Can software be developed once and then deployed on different platforms, in different architectures, and in different configurations?

Software Vendor Challenge 4 - Development organization system integration. Developers of software functionality must also consider the organizational context of their functionality and keep into account systems that need to interface with the functionality, such as licensing systems, content management systems, feedback systems, and support systems. As these systems are opening up their interfaces to the outside world, it becomes easier to interface with these systems directly, enabling automatic content updates, automatic feedback, automatic licensing and billing, etc. In parallel, developers must keep into account that feedback can be returned from functionality as it operates in the field [9]. Challenges in the software functionality context concern the development of software product management frameworks [8] and the development of feedback and development cycle models.

Example: Requirements Engineering in a vibrant ecosystem - Another project includes the study of the requirements prioritization process at the aforementioned Dutch software vendor, now that new participants can develop domain specific plug-ins that the software vendor does not consider core business. We are planning to study the way in which these requirements are communicated in the ecosystem and how these are picked up and implemented by partners.

Example: Architecting for Extensibility - As software vendors are opening up their interfaces and architectures, the decision needs to be made on how open architectures must be. Typically software vendors will only open up a small part of the architecture to protect intellectual property and to remain able to evolve the software without it becoming a maintenance nightmare. For this research several application program interfaces will be studied to establish architectural decisions made in regards to the SSN.

Example: Generating Feedback - Profiling software behavior in the field is frequently an afterthought for software developers. We are currently developing a software behavior logging tool, that applies aspect oriented programming techniques to easily weave logging code into any type of software product or service. The use of such tools enables software vendors to respond quickly to changes to a product or service in the field. Furthermore, a software vendor can monitor ongoing software usage in real-time [9].

5. Research Directions

We classify this research as empirical software engineering. Most of the challenges addressed in this paper are of a technical or business nature. Several economical issues are still open, such as a comprehensive comparison between traditional supply networks and software supply networks and surveys of large numbers of software related businesses and their financial results compared against their respective SECOs. Furthermore, more research must be done on concepts that can be copied from law, economics, and biology. We wish to gain support from colleagues in different fields to do interdisciplinary research.

References

[1] S. Brinkkemper, I. van Soest, and S. Jansen. Modeling of product software businesses: Investigation into industry product and channel typologies. In *proceedings of the Sixteenth International Conference on Information Systems Development*, pages 677–686. Springer-verlag, 2007.

[2] M. A. Cusumano. The changing software business: Moving from products to services. *IEEE Comp.*, 41(1):20–27, 2008.

[3] B. Farbey and A. Finkelstein. Exploiting software supply chain business architecture: a research agenda. In *proceedings of the 1st Workshop on Economics-Driven Software Engineering Research (EDSER-1)*, 1999.

[4] S. Jansen, S. Brinkkemper, I. Hunink, and C. Demir. Pragmatic and opportunistic reuse in two innovative start-up companies. *IEEE Software*, November/December, 2008.

[5] S. Jansen, A. Finkelstein, and S. Brinkkemper. Providing transparency in the business of software: A modelling technique for software supply networks. In *In Proceedings of the 8th IFIP Working Conference on Virtual Enterprises*, pages 677–686, 2007.

[6] D. G. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press, Cambridge, MA, USA, 2003.

[7] D. Postmus and T. D. Meijler. Aligning the economic modeling of software reuse with reuse practices. *Infor. Software Technology*, 50(7-8):753–762, 2008.

[8] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma. Towards a reference framework for software product management. pages 319–322, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[9] H. van der Schuur, S. Jansen, and S. Brinkkemper. Becoming responsive to service usage and performance changes by applying service feedback metrics to software maintenance. In *4th Intl. ERCIM Workshop on Software Evolution and Evolvability*, 2008.