Evolving Software Products, the Design of a Water-Related Modeling Software Ecosystem

Konstantinos Manikas[1,2(✉)]

[1] DHI Group, Hørsholm, Denmark
kman@dhigroup.com
[2] Computer Science Department, IT University of Copenhagen, Copenhagen, Denmark

Abstract. Software product evolution by means of improving their architecture, tools, or development methodologies are rather common in the lifetime of a software product. Especially if the product is in the domain of engineering where some of the basic calculation principles were established in some cases more than 50 years ago. However, a radical change of software products to evolve both in the software engineering as much as the organizational and business aspects in a disruptive manner are rather rare.

In this paper, we report on the transformation of one of the market leader product series in water-related calculation and modeling from a traditional business-as-usual series of products to an evolutionary software ecosystem. We do so by relying on existing concepts on software ecosystem analysis to analyze the future ecosystem. We report and elaborate on the main focus points necessary for this transition. We argue for the generalization of our focus points to the transition from traditional business-as-usual software products to software ecosystems.

Keywords: Software ecosystems · Ecosystem design · Product modernization

1 Introduction

Software ecosystems have been gaining in popularity in the past decade. We have noticed an increasing number of software systems and products being either converted from an existing system to an ecosystem or designed from the beginning to support the ecosystem approach [1]. The field of software ecosystems arguably appeared around the previous decade. Since then, the field has been shaped by a number of publications such as several literature studies [2–5] as much as a number of influential studies proposing, among other, means of analysis or categorizing software ecosystems [6–9]. Software ecosystems arguably come with several advantages such as increased level of innovation, better quality of software products, accelerated development, or reduced time to market. However not all

ecosystems have been equally successful or effective in achieving the advantages that are promised with this approach. This the ecosystem design with respect to success has been a focus of both research and academia.

In related work, [10] follow the transition of a software product line to a software ecosystem with the parallel transition from waterfall development processes to agile. [11] follow a similar transition of a proprietary platform but to an open source software ecosystem. In a somewhat different approach [12,13] describe the steps of analyzing and designing a software ecosystem around the telemedical services of the Danish healthcare. As noted form these studies, software ecosystems can mainly emerge from a successful software product (or system, company). However, the telemedicine ecosystem is an example the design of an ecosystem by identifying a need for an ecosystem rather than evolving from an existing platform. The steps towards the design of an ecosystem are elaborated more in [14]. One of the methods for analyzing and designing software ecosystems from a wide perspectives is the concept of "software ecosystem architecture" that we are using in this study [12]. The concept proposes that a software ecosystem can be analyzed by three different perspectives using three structures:

Software structure. That contains the different software elements of an ecosystem as much as their relationships (including software interaction). In some cases it helpful to separate between the common software infrastructure, i.e. the platform of the ecosystem, and the software extensions, i.e. contributions on top of the infrastructure.

Organizational structure. The different organizational elements of the ecosystem, such as the different actors that are involved in the ecosystem, their roles, and relationships.

Business structure. The different business elements of the ecosystem such as the incentives that motivate the actor activity in the ecosystem.

In this paper we focus on the design of a software ecosystem and report on the analysis of one of the market-leader suites of products in water-related modeling and prediction. Our case focuses in designing the evolution of the development and distribution from traditional means to a software ecosystem with utter aim to "solve the worlds toughest challenges in water environments" [15].

2 Approach

The studied systems are a set of software systems that, among other, perform complex calculation and modeling scenarios in a wide variety of water resource problems. These systems have been evolved and improved over the years with the first calculation algorithms dating back to the 1960's. The architecture of the different products includes high level or reuse and conceptual separation. The organization developing the systems is part of a wider non-profit organization of more than 1000 employees world-wide with main business water resources engineering expertise. The organization's products include project development and consultancy, software products, and knowledge distribution.

In the following section we discuss the result of analysis of the ecosystem to-be. We do so, in a generic manner so to be applicable in wider domains. We use the three structures of the software ecosystem architecture.

2.1 Software Structure

The transition to a software ecosystem poses a number of requirements to the software structure. This structure includes both the ecosystem *platform*, i.e. the software infrastructure that forms the core of the ecosystem where extensions are build upon, and the *software extensions*, i.e. plug-inns or apps that provide additional functionality or services to the ecosystem by extending the platform. Bellow we elaborate on the main aspects that the software structure of the ecosystem should cover.

Modularity and Independence. The clear separation of the (software and logical) components of an ecosystem is essential to the well-functioning and prosperity of the ecosystem. The better each logical (and thus software) entity is defined and separated from the rest, the more probable it is for the system to be to keep faithful to the architectural design. This is especially relevant to existing systems transitioning to an ecosystem as the effort of re-designing and refactoring is arguably greater than design from scratch. In such cases apart from re-designing, analysis of architectural evolution and specifically architectural drift and decay is very relevant as much as information on the initial architectural decisions and trade-offs. The proposed architectural smells [16] can be an relevant starting point. Today there is a number of architectural patterns and tactics that can facilitate this transitions, e.g. the use of service oriented architecture and microservices, as much as different tools.

Independent and Continuous Release. One positive effect of an optimal modularity in a system is that this logical structuring allows for releasing modules independently from each other. Independent release of modules is essential for the (rapid) evolution of large and complex systems as (a) it enforces complete control of dependencies - otherwise the system fails at runtime, (b) better supports the actor extension development as it allows the actors to focus only on the module(s) that are relevant, (c) allows for organizational independence, i.e. different organizations (or teams) can limit their scope easier. Apart from the platform, independent release and release roadmapping should also be a requirement for extensions that are reused by other components.

Standardization of Platform and Extension. In order to facilitate the rapid and proper extension development, the aspects of this development should be standardized to the extent possible. In that respect, the ecosystem should provide and enforce standardized means of development, deployment, and testing/quality assurance. Standardization could be included in following ways:

Documentation and Support. The ecosystem orchestrators should facilitate the ecosystem extension by "flattening" the learning curve of ecosystem contributors and establishing and communicating official "ways of doing". Examples here include good guides and documentation on how to create extension or standard functions e.g. user interface, logging, or error-handling. This should already by considered by design time by including relevant system architectural qualities, e.g. buildability.

Enforcement and Control. Some of the standardized procedures, might be imperative to be followed e.g. procedures dealing with authentication, authorization, or privacy. In those cases, apart from communication and support, there should also exist means of enforcement and control of the proper design and implementation. Depending on the governance the ecosystem is following, different ways of enforcement can be applied. Practices can vary from automated controls (e.g. during commit/deployment or binary controls), to manual and resource-demanding controls e.g. it is common approach to establish compliance and certification organizations or auditing procedures for systems with high requirements in quality assurance.

Coordination, Plan Communication, and Roadmapping. Development both internally in the platform and externally in the extensions can be rather distributed and independent. This can cause several issues related to the distributed work, e.g. extensions being build on a platform component/service that changed, or platform releasing similar functionality to what an external actor was building in an extension. This kind of issues can be arguably prevented by setting requirement for communicating changes and roadmaps of system evolution that other actors can align with. In some cases, a special organizational entity or automated system can be responsible for the communication and co-ordination of the software interaction.

2.2 Business Structure

The transition to a software ecosystem arguably has a great impact to the business structure. Below we elaborate on some of the aspects that should be covered:

2.3 Disruption and Business Development

Transitioning to an ecosystem potentially includes a disruption to the "business-as-usual" model that an organization might have established. Identifying the new business models and incentives both for the (orchestrating) organization itself but also for potential external organizations is essential for this transition. Aspects in this work include challenging the existing and identifying new: (i) value propositions, (ii) customer segments, (iii) revenue streams, (iv) strategic alliances.

Business and Software Structure Alignment. Similar to a single organization, the alignment of the business and the software is essential. Naturally, in the ecosystem perspective the complexity is of higher magnitude. The platform should reflect the business and the business should support the operation and evolution of the software structures. A proper set of value propositions and incentives both for the orchestrator - organization opening the platform and the software extension organizations is equally important (if not more) with a proper software structure. Means of designing the business structure moves towards traditional business development. Moreover the alignment of software and business structures can arguably be facilitated by theories and frameworks in the enterprise architecture. Naturally, these frameworks should also be extended to the ecosystem views.

2.4 Organizational Structure

Internal Organization. The transition to an ecosystem implies challenges to the internal organization that would take the role of the ecosystem orchestrator. A radical restructuring of the software and business in an organization should be followed by restructuring in the organization itself. A relevant example is, assuming that "Conway's law" [17] is valid, the platform would reflect the structure of the organization and the pattern of communication. Thus, the structure of the organization should be evaluated in this light. Moreover, the software and business structure aliment should also be reflected here.

External Organization. The structuring of the external to the orchestrator organizations could potentially include implications that need to be addressed.

Actor Involvement Model. How external actors are to be included is an important aspect on an ecosystem. If the ecosystem is very open to external actors/organizations, there might appear issues with high extension competition that might have a negative effect to the ecosystem. Moreover, the more the ecosystem contributions scale, the more challenging it might be to control and maintain quality. On the other hand, if an ecosystem is to limiting to external actor inclusion, the ecosystem might not be able to obtain and maintain a critical mass for the ecosystem to evolve and eventually survive.

Defining Internal and External. Transition to an ecosystem also implies that external organizations might be occupied with aspects of a systems that was previously internal. It might be necessary to define and make explicit the borders of each system and the responsibilities of each actor in a more formal way to avoid organizational and legal frictions. Moreover, in cases of privacy and risk of leak of important information, employees should have guidance on the right level of communication and the privacy level of information. This is something that is implemented in many organizations today. The challenge increases with the increase in complexity, e.g. more actors in different privacy levels.

3 Conclusion and Future Work

In this paper we report on the transition of a software product suite to a software ecosystem. We rely on the concept of software ecosystem architecture and analyze the current systems. Our work results in a set of focus points that are necessary for the transformation to an arguably healthy ecosystem.

Plans for future work include the evaluation of the focus points and the detailed design of the aspects that the focus points identify. We argue that the identified points can be developed further to a generalized method for evolving from traditional software systems to software ecosystems.

References

1. Manikas, K.: Supporting the evolution of research in software ecosystems: reviewing the empirical literature. In: Maglyas, A., Lamprecht, A.-L. (eds.) Software Business. LNBIP, vol. 240, pp. 63–78. Springer, Cham (2016). doi:10.1007/978-3-319-40515-5_5
2. Hanssen, G.K., Dybå, T.: Theoretical foundations of software ecosystems. In: Jansen, S., Bosch, J., Alves, C. (eds.) Proceedings of the Forth International Workshop on Software Ecosystems, Cambridge, vol. 879, pp. 6–17, 18 June 2012. http://CEUR-WS.org
3. Barbosa, O., Santos, R.P., Alves, C., Werner, C., Jansen, S.: In: Software Ecosystems - Analyzing and Managing Business Networks in the Software Industry. Edward Elgar, Cheltenham (2013)
4. Manikas, K., Hansen, K.M.: Software ecosystems - a systematic literature review. J. Syst. Softw. **86**(5), 1294–1306 (2013)
5. Manikas, K.: Revisiting software ecosystems research: a longitudinal literature study. Syst. Softw. **117**, 84–103 (2016)
6. Bosch, J.: From software product lines to software ecosystems. In: Proceedings of the 13th International Software Product Line Conference SPLC 2009. Carnegie Mellon University, Pittsburgh, pp. 111–119 (2009)
7. Jansen, S., Finkelstein, A., Brinkkemper, S.: A sense of community: a research agenda for software ecosystems. In: 31st International Conference on Software Engineering - Companion, vol. 2009, pp. 187–190. ICSE-Companion, May 2009
8. Knodel, J., Manikas, K.: Towards a typification of software ecosystems. In: Fernandes, J.M., Machado, R.J., Wnuk, K. (eds.) ICSOB 2015. LNBIP, vol. 210, pp. 60–65. Springer, Cham (2015). doi:10.1007/978-3-319-19593-3_5
9. Manikas, K., Hansen, K.M.: Reviewing the health of software ecosystems - a conceptual framework proposal. In: Proceedings of the 5th International Workshop on Software Ecosystems, Potsdam, vol. 987, pp. 33–44, 11 June 2013. http://CEUR-WS.org
10. Hanssen, G.K.: A longitudinal case study of an emerging software ecosystem: implications for practice and theory. J. Syst. Softw. **85**(7), 1455–1466 (2011)
11. Kilamo, T., Hammouda, I., Mikkonen, T., Aaltonen, T.: From proprietary to open source-growing an open source ecosystem. J. Syst. Softw. **85**(7), 1467–1478 (2012)
12. Christensen, H.B., Hansen, K.M., Kyng, M., Manikas, K.: Analysis and design of software ecosystem architectures - towards the 4s telemedicine ecosystem. Inf. Softw. Technol. **56**(11), 1476–1492 (2014)

13. Manikas, K.: Analyzing, Modelling, and Designing Software Ecosystems - Towards the Danish Telemedicine Software Ecosystem. PhD thesis, Department of Computer Science, University of Copenhagen, Denmark (2015)
14. Manikas, K., Hämäläinen, M., Tyrväinen, P.: Designing, developing, and implementing software ecosystems: towards a step-wise guide. In: The 8th International Workshop on Software Ecosystems (2016)
15. DHI Group: Our foundamentals. Accessed 23 Feb 2017. https://www.dhigroup.com/about-us/corporate-social-responsibility/our-fundamentals
16. Garcia, J., Popescu, D., Edwards, G., Medvidovic, N.: Identifying architectural bad smells. In: 2009 13th European Conference on Software Maintenance and Reengineering, pp. 255–258, March 2009
17. Conway, M.E.: How do committees invent. Datamation **14**(4), 28–31 (1968)