On the Importance of Sound Architectural Practices in the Use of OSS in Software Product Lines

Muhammad Ali Babar[1], Brian Fitzgerald[1], Pär J. Ågerfalk[12], Björn Lundell[3]

[1]Lero, University of Limerick, [2]Uppsala University
[3]University of Skövde

Abstract. Software Product Line (SPL) and Open Source Software (OSS) have emerged as successful modes of developing software. Although they appear to differ in terms of development principles and processes, researchers and practitioners have been increasingly emphasising the need to achieve synergies by exploiting the ever growing repositories of OSS components for developing SPLs. While there have been calls for the SPL community to accelerate the widespread use of OSS in SPL, less attention has been paid to how OSS communities could increase the use of OSS components in SPL. Since architectural issues are considered critical in the SPL community, we propose that an increased attention on architectural aspects of OSS components may provide the confidence that organizations need in order for them to choose and use OSS components in SPL. We identify a number of architectural practices which are followed by the SPL community and discuss the possibilities for and potential benefits of incorporating those practices in OSS development processes.

1. Introduction

Software product line (SPL) and Open Source Software (OSS) have received broad recognition as emerging modes of software engineering with the potential to revolutionize the principles and practices of software development. Since their origin, both approaches have been perceived to be entirely different in terms of principles and practices [1]. The SPL approach promises to help organizations maximize the intra-organizational reuse of software artefacts in order to reduce cost, improve time-to-market, and achieve better quality [2]. To realise the objective of large-scale reuse of artefacts, the SPL approach focuses on disciplined processes, heavy documentation, and rigorous design and assessment of software architectures [3]. The OSS movement, which originated from a pragmatic need to share code among individuals has grown to become a major force behind inter-organizational reuse of platforms, components and code. Commercial and collaborative opportunities for inter-organizational reuse are seen as the main driving force behind the changing landscape of the OSS paradigm [4]. Researchers and practitioners have been exploring the opportunities and challenges of utilizing the ever growing repositories of shared components provided by OSS in software product lines. The use of OSS components in SPL appears to have great potential for both the OSS and SPL communities. For the SPL community, the use of OSS components in a SPL promises to help them to minimize the development efforts in commodity (non-value adding)

components. Several OSS components have been successfully used in mission-critical product families [5]. Several researchers have also been encouraging the SPL community to follow the practices and principles of software development found effective and efficient in OSS projects. It is claimed that the SPL community can greatly benefit from learning and adopting certain OSS practices and principles such as communication mechanisms, tooling support, code ownership, technical roadmap, quality management, and release management [1]. Whereas we acknowledge the importance of analysing the OSS development processes and practices for their suitability in the SPL development processes, we contend that it is also useful to explore and highlight the aspects of the OSS development processes that may need to be adjusted in order to help make the use of OSS components in SPL widespread.

Hence, we argue that there is a need to emphasise the critical role that an OSS community can play in increasing the use of OSS components used in SPL development projects. One way of helping organizations to confidently choose and use OSS components in their core assets as well as in customized products is to pay more attention to the architectural aspects during the development and evolution of those components. While it is a well-known fact that the success of OSS projects tends to rely on good and solid software architectures [6], the role of architecture in OSS needs further elaboration [7] and it appears that the architectural design process lacks transparency and is poorly documented.

Software architecture and its related issues are considered of paramount importance in the successful development and maintenance of a SPL [2, 3]. Software architecture is an effective mechanism of reducing complexity and cost of developing and maintaining code, but also streamlining the production of documentation and training material [8]. Architectural mismatches between components from different developers have been found to be the major cause of integration mayhems [9, 10]. A rigorously designed and well-implemented architecture is expected to provide several benefits such as agility in response to volatile markets and emerging business requirements, improved time-to-market for new products, better quality, just-in-time reconstruction of legacy systems for changing requirements, and effective and efficient management and enhancement of many product variations needed for international markets [8]. That is why organizations heavily invest in software product-line architectures. Hence, many organizations feel nervous about the idea of using components whose development processes do not pay sufficient attention to the architectural aspects by rigorously assessing and explicitly documenting key architectural design decisions. Moreover, components integration efforts usually face problems without access to the contextual information about the design decisions and assumptions about the kind of environment or architectures suitable for integrating those components [11, 12].

In this position paper, we identify a number of challenges of using components where the architectural decisions and rationale underpinning those decisions are unknown. We also explore the literature of software product-line architecture to identify a few principles and practices of designing, assessing, and documenting architectures. Some discussion is also provided on how architectural practices of the SPL community can be incorporated in OSS development. This paper proposes that by following sound architectural practices and principles, OSS communities can promote the widespread of use of OSS components in SPL.

2. Background and Motivation

In this section we discuss the main concepts of SPL and software architecture practices in an effort to help the reader to understand the importance of sound architectural practices that can stimulate synergies between OSS and SPL.

2.1 Software Product Lines

A SPL is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [13]. The SPL approach makes a distinction between domain engineering, where a common platform for an arbitrary number of products is designed and realized, and application engineering, where a product is derived [14]. The separation into domain engineering and application engineering allows the development of software artefacts which are shared among the products within that domain. These shared artefacts become separate entities in their own right, subscribing to providing shared functionality across multiple products. It is during application engineering that the individual products within a product line are constructed. The products are constructed using a number of shared software artefacts created during domain engineering. The process of creating these individual products using the shared artefacts is known as the Product Derivation process, which can utilize both in-house developed and OSS components along with the core assets [2]. OSS components can also be used in developing core assets of a SPL. For example, TAO/CIAO have been used to build middleware for several SPLs in embedded system domain [5].

2.2 Software Architectures

Software architecture is one of the most important initial design artifacts that can be used as a roadmap to successfully develop software applications. It has been shown that software architecture is an effective tool to cut development cost and time and to increase the quality of a system [15]. The software architecture process consists of several activities (such as design, documentation, and evaluation), which involve complex knowledge intensive tasks [16, 17]. The knowledge that is required to make suitable architectural choices and to rigorously assess those choices is broad, complex, and evolving. Such knowledge is often beyond the capabilities of any single architect. The software architecture community has developed several methods (such as a general model of software architecture design [18], Architecture Tradeoff Analysis Method (ATAM) [19], and architecture-based development [20]) to support a disciplined architecture process.

A software architecture-based approach focuses on analyzing and evaluating the architectures of the existing component through architectural documentation or architectural archaeology, identifying any mismatch among those components, designing, documenting, disseminating, and maintaining the architectural description, reasoning about the detailed design in the context of high level architecture, and so forth [21-23]. Apart from these generic methods to support rigorous architecture design and evaluation, there are methods and techniques specifically developed for SPL [24]. The SEI's software architecture initiative under SPL umbrella has also

produced a family of design and evaluation methods, which incorporate ten fundamental techniques and three principles [25]. Dikel et al. have also reported six principles of applying product-line architecture in Nortel and lesson learned. These principles include focusing on simplification, adapting to future needs, establishing architectural rhythm, partnering with stakeholders, maintaining vision and managing risks and opportunities [8].

3. Software Architecture Practices in SPLs VS. OSS

The software architecture of the product line is the artefact that defines the overall decomposition of the products into the main components. A software product line architecture identifies the commonality among different products and helps explicitly document variability. The Product line architectures can be at different levels of maturity in their lifecycle. Bosch has identified three levels of architectural maturity in SPL organizations: under-specified architecture, specified architecture, forced architecture [11]. Software product line architecture, irrespective of its maturity level, is designed and assessed for functional and non-functional requirements of a particular domain using different methods and principles as described in the previous section. Software product line architectures are also rigorously documented and maintained using various architecture description languages, and documentation approaches such as the IEEE 1471 standards [26] and the Views and Beyond approach [27]. Both of these approaches also identify the rationale as an important part of architectural description and emphasize the importance of accumulating and managing architecture development experiences, which can play a key role in improving software architecture processes.

OSS communities present a different attitude towards software development. In principle, releasing the source code under an open source licence allows anyone to inspect and modify the code. The lack of explicit planning and formal project management in many open source projects puts additional strain on the architecture [7]. Most of the successful OSS projects have been in well-understood domains where requirements and architectural issues have matured. For example, Linux community has greatly benefited from the requirements and architecture defects that had already been found and fixed in many previous generations of Unix [28]. Additionally, majority of the successful OSS projects are focused on the infrastructural support systems such as Apache, Mozilla, and MySQL [4] whose requirements are relatively well-understood. That can be one of the reasons that OSS project initiators would rarely feel it valuable to have a lengthy phase of requirements elicitation and specification; nor would they be interested in rigorously describing and continuously updating architectural design decisions. It is therefore not uncommon for OSS projects to downplay the importance of an explicit design activity with proper design documentation. However, this does not necessarily mean that such projects lack architectural design and requirements [1]. Instead, unlike SPL, it is quite rare to find use cases or scenarios for specifying architectural requirements for OSS projects. Rather, projects typically have detailed requirements and change requests managed through issue and bug tracking systems, and also use other light-weight development tools typically offered on Internet-based development platforms.

Since OSS is not developed to fulfil the requirements of a particular set of products, it is highly likely that SPL developers would find fundamental differences between the OSS and SPL approaches. Confusion can also arise due to differing assumptions about the architectural environment for which an OSS component was developed. Also interfaces provided by OSS components and those required by an SPL may differ. Likewise, the quality attributes required by an SPL and those provided by an OSS component may diverge. The software architecture community follows various design principles to address the potential architecture mismatch challenges in components integration. One of those practices is rigorously documenting and widely publishing component's interfaces and explicating architectural and environmental assumptions. Wiki-based communication tools widely used by OSS communities can be used for this purpose. Due to the traditionally "bazaar-like" phenomenon of OSS development, insufficient consideration was paid to various aspects of sound design of a component, which may result in design erosion over time. Instead, OSS communities adopts the agile principle of regularly re-factoring already implemented components [28]. However, there is an element of risk for a SPL user of such components, as design erosion from the evolution of one component may have a ripple effect not only across core components of a SPL but also across the whole set of products that belong to a particular SPL. This issue can result in increased cost of maintenance, complexities in the evolution of core assets as well as individual products, or abandonment of that SPL.

Choosing and using components based on a product-line architecture require considerable knowledge of the rationale and concepts underlying the reusable components. Rather than just knowing a component's interface, it is important to know about the architecture for which the component was developed, the semantics of the behaviour of the component and the quality attributes for which the component has been optimized. Such information is unlikely to be available if component developers, irrespective of proprietary or OSS, are not following a rigorous and disciplined architecture process. Additionally, a lack of effective documentation practices in OSS communities can make it a challenge to fully analyze and understand an OSS component for its compatibility with a product line architecture and requirements.

4. Conclusions

This paper reports on an ongoing investigation into the challenges and opportunities in facilitating the use of OSS components in developing software product lines. We believe that by exploiting the increasing amount of high quality components provided by OSS communities, several benefits can be achieved by both OSS and SPL communities. We position our research along the lines of other efforts aimed at bridging the gap between OSS and SPL development paradigms to create new synergies. We argue that an increased attention to architecture processes can play an important role in accelerating the use of OSS components in the SPL. Hence, while the SPL developers can benefit from learning and adopting the processes and practices found effective and efficient in OSS projects, there is also an increasing need for treating architecture as a first class artefact in OSS projects by rigorously and

explicitly documenting architectural design decisions and contextual information surrounding those decisions.

We acknowledge that many OSS projects produce effective architectural designs. However, if OSS project design processes are not transparent and the software architecture is poorly documented and justified, this may prove to be the biggest stumbling block in the widespread adoption and deployment of OSS components by the SPL community. In this paper, we have identified several methods and principles that are usually followed by the SPL community to design and maintain product-line architectures. While some of these methods and principles may be very specific to proprietary software development, there are many of them that can be incorporated in the OSS development processes to make architecture a first class artefact. In fact, there are many examples of good design principles, i.e., focusing on simplifications and modularization, which usually drive the architecture design decisions in OSS systems. However, in many cases the architectural design decisions are not explicitly documented or made available to users of components, which may pose risks in using those components as identified in the previous section.

Moreover, we propose that a possibly utopian but yet promising way of bridging the gap between open source communities and the SPL community may be to organize OSS projects along the patterns of organising SPL development; i.e. development department, business units, domain engineering, and hierarchical domain engineering [29]. Since, it is difficult for a green-field OSS project to anticipate its potential for growing into a family of products, existing successful and popular projects can certainly be analyzed for SPL organizations. For such analysis, the SPL community has developed several methods and techniques that can be used to reconstruct architectures of OSS projects for evaluating and refactoring purposes. There are also various techniques available to support the product line potential analysis, which can help OSS project owners and communities to quickly identify the challenges and opportunities that may exist to reorganize a project as an SPL. However, it may be difficult to organize and motivate developers to participate in an open source project that has adopted (what may perceived as) too rigid SPL practices. Hence, the feasibility of such an approach is open for enquiry. This, then, suggests a need for further research on exploring what SPL practices may be suitable to adopt, without distorting the open source 'ecosystem'.

7. References

[1] J.V. Gurp, OSS Product Family Engineering, in The first International Workshop on Open Source Software and Product Lines. 2006.

[2] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. 2001: Addison-Wesley.

[3] J. Bosch, Design & Use of Software Architectures: Adopting and evolving a product-line approach. 2000: Addison-Wesley.

[4] B. Fitzgerald, The Transformation of Open Source Software *MIS Quarterly*, 2006. 30(3).

[5] D. Schmidt, Model Driven Engineering of Product-Line Architectures for Distributed Real-time and Embedded Systems, *Tech Report* Vanderbilt University, USA, 2007.

[6] J.Y. Moon and L. Sproull, Essence of Distributed Work: The Case of the Linux Kernel, *First Monday*, 2000. 11(5).
[7] B. Arief, C. Gacek, and T. Lawrie, Software Architectures and Open Source Software – Where can Research Leverage the Most?, *The 1st workshop on Open Source Software Engineering*, 2001.
[8] D. Dikel, et al., Applying Software Product-Line Architecture, *IEEE Computer*, August, 1999: pp. 49-55.
[9] D. Garlan, R. Allen, and J. Ockerbloom, Architectural mismatches, or Why it's hard to build systems out of existing parts, in The 17th International Conference on Software Engineering. 1995.
[10] M. Ali-Babar and D. Zwoghi, Developing a Requirements Management Toolset: Lessons Learned, *The 15th Australian Software Engineering Conference*, 2004.
[11] J. Bosch, Product-Line Architectures in Industry: A Case Study, in The 20th International Conference on Software Engineering. 1999.
[12] M. Ali-Babar and I. Gorton, A Tool for Managing Software Architecure Knowledge, *Proceedings of the 2nd Workshop on SHAring and Reusing architectural knowledge - Architecture, rationale, and Design Intent (SHARK/ADI 2007), Collocated with ICSE 2007.*
[13] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. 2002, Reading, MA: Addison Wesley.
[14] L. Hotz, A. Gunter, and T. Krebs, A Knowledge-based Product Derivation Process and some Ideas how to Integrate Product Development, in Proc. of Software Variability Management Workshop. 2003: Groningen, The Netherlands.
[15] L. Bass, P. Clement, and R. Kazman, Software Architecture in Practice. 1998, Reading, Massachusetts: Addison-Wesley.
[16] P.N. Robillard, The role of knowledge in software development, *Communication of the ACM*, 1999. 42(1): pp. 87-92.
[17] L.G. Terveen, P.G. Selfridge, and M.D. Long, Living Design Memory: Framework, Implementation, Lessons Learned, *Human-Computer Interaction*, 1995. 10(1): pp. 1-37.
[18] C. Hofmeister, et al., A General Model of Software Architecture Design Derived from Five Industrial Approaches, *Journal of Systems and Software, Article in the press*, 2006.
[19] P. Clements, R. Kazman, and M. Klein, Evaluating Software Architectures: Methods and Case Studies. 2002: Addison-Wesley.
[20] L. Bass and R. Kazman, Architecture-Based Development, *Tech Report CMU/SEI-99-TR-007*, Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, USA, 1999.
[21] D. Garlan, R. Allen, and J. Ockerbloom, Architectual Mismatch, or, Why it's hard to build systems out of existing parts, *17th International Conference on Software Engineering*, Apr.1995.
[22] L. Bass and R. Kazman, Architecture-Based Development, *Tech Report CMU/SEI-99-TR-007*, CMU/SEI-99-TR-007,SEI, Carnegie Mellon University, Apr.1999.
[23] N. Medvidovic, D. Rosenblum, and R. Taylor, An Architecture-Based Approach to Software Evolution, *IWPSE*, Apr.1999.
[24] M. Matinlassi, E. Niemela, and L. Dobrica, Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture, *Tech Report 456*, VTT Technical Research Centre of Finland, Espoo, 2002.

[25] R. Kazman, L. Bass, and M. Klein, The essential components of software architecture design and analysis, *Journal of Systems and Software*, 2006. 79: pp. 1207-1216.
[26] IEEE, ed. Recommended Practices for Architecture Description of Software-Intensive Systems. 2000, IEEE Std 1471-2000.
[27] P. Clements, et al., Documenting Software Architectures: Views and Beyond. 2002: Addison-Wesley.
[28] S. McConnell, Open-Source Methodology: Ready for Prime Time?, *IEEE Software*, 1999. 16(4): pp. 6-8.
[29] J. Bosch, Architecture-centric software engineering, *Proc. the 24rd International Conference on Software Engineering*, 2002.