The Journal of Systems and Software 91 (2014) 1–2

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Editorial

Variability in software architecture – State of the art

1. Introduction

Today's software systems must cope with versatile environments, different user groups, and varying usage scenarios and deployment settings. As technology progresses and stakeholder requirements become more and more difficult to predict, usage scenarios are often not known at early design or change during the system's lifetime. Therefore, important design decisions are postponed to a later stage when the software is adapted for a particular context. Consequently, the implementation of a software system can differ depending on the users or the domains in which the software is used. For example, an enterprise resource planning software may run in an oil and gas company or at a university. Both organizations would have similar but differing requirements (for instance, both organizations need Human Resource (HR) functionality, but how HR is managed differs between a privately operated oil and gas company and a publicly funded university). Thus, instead of building completely independent software for both organizations, one system may be built that can be adapted for either oil and gas or university domains through *variation points* in the design, and *variants* to resolve variation points. This approach has proven to significantly reduce development and maintenance effort. Therefore, many of today's software systems are built with variability in mind, such as software product lines or families, self-adaptive systems, and open platforms.

The architecture of a software-intensive system is the reference point for all development activities and the earliest point where significant design decisions are taken. Furthermore, the software architecture plays a significant role for achieving quality attributes, such as security, performance, or reliability (Bass et al., 2003). If variability is not taken into account during architecture design, there is a risk of compromising quality attributes and increasing later rework. Therefore, recent community efforts have identified the need to treat variability as a first-class concern throughout software architecture design (Galster et al., 2011, 2013). This poses new challenges on the design, implementation and quality assurance of software-intensive systems. Some research challenges related to variability in software architecture include the following:

- Variability not only affects functionality but also the quality of software systems. For example, variability in quality attributes can occur due to variations in different market segments or customer profiles (e.g., premium customers may get better performance compared to regular customers). In this scenario, performance requirements are "variation points".
- We may not be able to predict all possible variants at design time since the complexity of today's software systems and the behavior of users makes it impossible to predict all usage scenarios. This requires adaptation at runtime and deferring the realization of variability from design time to runtime.
- In embedded systems, variability can either be handled in the software or the hardware part. The software architecture must consider the resource constraints in embedded systems when implementing variability. For example, it may not be feasible to implement all possible variants in the software of an embedded system due to memory constraints.
- The software architecture documentation is organized into different models and architecture views. Thus, architects must be able to model variability in and across different architecture models or views (e.g., logical or process views). At the same time we must ensure consistency and traceability of variability from requirements to architecture to implementation.

In this special issue we want to highlight the importance of handling variability at the architecture level when building complex software systems that meet the needs of different stakeholders. Thus, this special issue aims at providing researchers and industry professionals with insights into the state-of-the-art in variability at the software architecture level. All submitted papers went through a rigorous multi-staged review process. From the 20 submissions, we accepted four high-quality papers for inclusion in the special issue.

2. Overview of the issue

The four articles that appear in this special issue attempt to address some of the research challenges stated before. The concept of variability in software architecture is fairly broad and the articles in this special issue provide different perspectives within this breadth. On the one end of the spectrum, this special issue includes contributions on extending traditional software product line engineering toward dynamic variability and dynamic product line architectures. On the other end of the spectrum, we include contributions related to particular software engineering activities (e.g., software testing). The domains of Service-Oriented Architecture and Self-Adaptation are also quite prominently represented in this special issue, as they provide significant technologies for handling variability.

As argued above, one current research challenge is variability at runtime. In their paper, Capilla et al. provide a view on runtime variability in the architecture of dynamic software product lines. Based on observations from research and industrial practice in embedded systems, ecosystems, service-based applications and self-adaptive systems, the authors explain how a software architecture can be

0164-1212/$ – see front matter
http://dx.doi.org/10.1016/j.jss.2014.01.051

designed to support dynamic adaptation and reconfiguration. The authors first provide an informal overview of dynamic software product lines and explain differences between a dynamic product line and "conventional" product lines. This overview includes a view of challenges and solutions that occur in the context of dynamic product line architectures. Then, based on this overview, the authors characterize the main parts of a dynamic product line architecture, including mechanisms that any dynamic software product line should support in order to facilitate dynamic variability and adaptation. Finally, the authors illustrate their concepts and ideas with examples of application domains in which a dynamic product line architecture may succeed.

Service-oriented architecture and service-based development has become a widely studied and used concept in software engineering research and practice (Dodani, 2006; Murer and Hagen, 2014). The paper of Alféreza et al. deals with the timely and significant challenge of dynamic adaptation of service compositions by using models at runtime. In the presence of events that can threaten the overall quality of the composition (or the corresponding SLA), there is a need for automatic (run-time) decision making to reconfigure the service composition. To this aim, the authors present a feature-based variability model, which abstracts away service compositions as sets of features. Similarly to selecting features in a product line at design time, this technology-independent model can be used to select variants of compositions at run time which are directly translated to the running service composition by adding or removing fragments of WS-BPEL code. In addition, the variability model and its configurations are verified at design time to ensure that selected configurations at run-time will still obey system and quality constraints. The proposed solution is empirically evaluated and evidence is presented to support the applicability and benefits, both at design time and at runtime.

More and more software systems are created by assembling software services that are provided by software vendors when needed (Turner et al., 2003). Vendors offer services and customer organizations use these services to reduce the effort when developing systems. Based on this idea, Walraven et al. present an integrated service engineering method, called service line engineering, that facilitates the development and management of customizable, multi-tenant Software-as-a-Service (SaaS) applications, without compromising scalability. The method spans the design, implementation, configuration, composition, operation and maintenance of a SaaS application that bundles all variations that are based on a common core. The authors illustrate the benefits of the method in the development of a real-world SaaS offering for document processing. The evaluation shows that the effort to configure and compose an application variant for each individual tenant is significantly reduced, though at the expense of a higher initial development effort. The authors emphasize the need for studies on Cloud computing from a software engineering perspective, in particular regarding multi-tenancy at the application level (SaaS), and position their work as an initial exploration to deal with this need.

A recent systematic literature review on variability in software systems found that variability has been researched in all software engineering phases (requirements engineering, implementation, etc.) but that testing is underrepresented (Galster et al., 2014). The work of Lity et al. is one attempt toward filling this gap. Lity et al. propose an architecture-based testing approach for large-scale systems by combining principles from model-based testing and regression testing. The approach is based on delta-oriented architectural modeling, including the decomposition of a system into components and connectors, use of state charts for modeling component internal behavior and message sequence charts for inter-component behavior. One important expected benefit from use of the approach is a reduction of the testing effort due to the reuse of test artifacts. The approach has a solid formal basis, which provides additional benefits, such as guarantees on stable test coverage over the varying systems. In addition to building for the formal basis, the researchers link their work to the practice by evaluating the approach with an industrial case from automotive industry.

In conclusion, this special issue on variability in software architecture addresses a critical area of software architecture research and practice. The selected articles address different perspectives but complement each other by providing views from software architecture practice but also well-accepted software engineering paradigms. We appeal to the software architecture community to continue the work on this research area as the needs for variability will only grow in the future, and the more we learn the more open questions we come across (Galster et al., 2014). This guarantees both challenging and rewarding times ahead.

Acknowledgments

We thank all the authors for submitting their work to the special issue. We are grateful to the reviewers for their excellent work. Finally, we thank the editor-in-chief for his support throughout the process of preparing this special issue, and JSS for hosting this special issue.

References

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice. Addison-Wesley, Boston, MA.
Dodani, M.H., 2006. SOA 2006: State of the Art. Journal of Object Technology 5, 41–48.
Galster, M., Avgeriou, P., Weyns, D., Männistö, T., 2011. Variability in Software Architecture: Current Practice and Challenges. ACM SIGSOFT Software Engineering Notes 46, 30–32.
Galster, M., Weyns, D., Avgeriou, P., Becker, M., 2013. Variability in Software Architecture: Views and Beyond. ACM SIGSOFT Software Engineering Notes 38, 46–49.
Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P., 2014. Variability in software systems - A Systematic Literature Review. IEEE Transactions on Software Engineering 91, 1–2.
Murer, S., Hagen, C., 2014. 15 years of service oriented architecture at credit suisse. IEEE Software (in press).
Turner, M., Budgen, D., Brereton, P., 2003. Turning software into a service. IEEE Computer 36, 38–44.

Matthias Galster is a lecturer in Software Engineering at the University of Canterbury in Christchurch, New Zealand.

Paris Avgeriou is a professor and head of the Software Engineering and Architecture group at the University of Groningen, Netherlands.

Tomi Männistö is a professor at University of Helsinki, Finland.

Danny Weyns is a professor at Linnaeus University, Sweden.

Matthias Galster*
University of Canterbury, New Zealand

Paris Avgeriou
University of Groningen, The Netherlands

Tomi Männistö
University of Helsinki, Finland

Danny Weyns
Linnaeus University, Sweden

* Corresponding author. Tel.: +64 3 364 2347.
E-mail addresses: mgalster@ieee.org (M. Galster), paris@cs.rug.nl (P. Avgeriou), tomi.mannisto@cs.helsinki.fi (T. Männistö), danny.weyns@lnu.se (D. Weyns)

25 January 2014
Available online 3 February 2014