Variability in Software Architecture – Views and Beyond

Matthias Galster
University of Groningen, The Netherlands
mgalster@ieee.org

Danny Weyns
Linnaeus University, Sweden
danny.weyns@lnu.se

Paris Avgeriou
University of Groningen, The Netherlands
paris@cs.rug.nl

Martin Becker
Fraunhofer IESE, Germany
martin.becker@fraunhofer.iese.de
DOI: 10.1145/2382756.2382768
http://doi.acm.org/10.1145/2382756.2382768

ABSTRACT
Variability (the ability of a software system or software artifact to be adapted for use in a specific context) is reflected in and facilitated through the software architecture. The Second International Workshop on Variability in Software Architecture (VARSA) was held in conjunction with the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture 2012 in Helsinki, Finland. The workshop aimed at exploring current and emerging methods, languages, notations, technologies and tools to model, implement, and manage variability in the software architecture. It featured one industrial talk, five research paper presentations, and three working group discussions. Working groups discussed topics that emerged during the workshop. This report summarizes the themes of the workshop and presents the results of the working group discussions.

Categories and Subject Descriptors
D.2.10 [**Software Engineering**]: Design – *methodologies*. D.2.11 [**Software Engineering**]: Software Architectures – *data abstraction*. K.6.3 [**Management of Computing and Information Systems**]: Software Management – *software development*.

Keywords
Variability, software architecture, VARSA.

5. INTRODUCTION
Variability is the ability of a software system or software artifact to be adapted for specific contexts, in a preplanned manner. Many of today's software systems are built with variability in mind, e.g., product lines or families, self-adaptive systems, open platforms, or service-based systems with dynamic runtime composition of services. Variability is reflected in and facilitated through the software architecture. Thus, variability should be treated as a first-class and cross-cutting concern in software architecture. Also, the software architecture is a reference point for many development activities (detailed design, implementation, maintenance, etc.) and for achieving quality attributes (e.g., performance, security, and reliability). In this context, the Second International Workshop on Variability in Software Architecture (VARSA 2012) aimed at exploring current and emerging methods, languages, notations, technologies and tools to model, implement, and manage variability in the software architecture.

VARSA 2012 was held in conjunction with the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture 2012 (WICSA/ECSA 2012) in Helsinki, Finland. Around 35 participants were registered for the workshop. Further information about the workshop, its theme, motivation, etc. can be found in the WICSA 2012 proceedings [1] as well as on the workshop website (sites.google.com/site/varsa2012/). The workshop was a follow-up event of the First International Workshop on Variability in Software Architecture (VARSA 2011), held at WICSA 2011 in Boulder, Colorado. The workshop report of VARSA 2011 can be found in [2].

6. WORKING GROUP DISCUSSIONS
The workshop accepted one industry paper and five research papers for inclusion in the proceedings. The papers were presented in 20 minutes presentations. For details about accepted papers please refer to [1]. The presentations provided starting points for the discussion in three working group sessions. The following topics were selected for further discussion:

1) Variability at runtime: Variability at runtime is concerned with defining and implementing variability to be resolved during runtime, rather than at design time.
2) Reference architectures versus software product line architectures: Reference architectures and product line architectures are similar, yet there are differences between them.
3) Traceability of architectural variability: Variability should be traced across and within different architecture descriptions (e.g., models, views).

The topics were selected based on the interests of workshop participants, i.e., the selected three topics received the most votes from the participants. Interestingly, all discussion groups utilized the concept of architectural viewpoints and views as the means for architecture description [3] when elaborating on the three topics.

We formed groups that established a balance between participants from academia and industry. Thus, all groups discussed both, the industrial and academic perspectives on the topics listed above. The following sections elaborate on the results of the discussions in the working groups.

6.1 Variability at Runtime

Variability at runtime (or runtime variability) is about resolving variability at runtime, rather than during the design of a software system. This means, runtime variability involves selecting and binding variants at a variation point at runtime, rather than at design time before a system is deployed. The group first brainstormed about topics that are relevant in the context of variability at runtime and that require further investigation. The following three topics were identified:

1) What are constraints that affect the amount of runtime variability that is required (or that is feasible)?
2) What are challenges or current needs related to variability at runtime?
3) What are potential solutions to enable variability at runtime?

Constraints that affect the amount of runtime variability. The group concluded that the amount of runtime variability that is required or that is feasible would depend on a number of constraints.

First, runtime variability requires that software is delivered to a customer with all possible variation points and variants implemented in the code and the executable / binary. However, this may be difficult to do when memory in a deployment environment is constrained (e.g., for embedded systems). When memory and other runtime resources are constrained, the delivered and deployed software should be as lean as possible and only the most important variants should be included in a deliverable.

Second, architecturally significant quality attributes (such as performance) may limit the amount of runtime variability that is allowed in a system. For example, choosing from a large number of variants during runtime and binding a variant may reduce performance. Thus, technical constraints and quality attributes that exist in a given environment or development project limit the amount of runtime variability that is feasible.

Third, the cost involved in implementing runtime variability and the additional complexity may not be justified by the benefit of runtime variability. For example, including all possible variants in a released software product increases flexibility at runtime but on the other hand increases testing and maintenance costs and efforts.

Fourth, the degree of distribution of a software system may limit the amount of runtime variability. In highly distributed systems (e.g., service-based applications) where variability is resolved at runtime (e.g., through dynamic service retrieval and binding), allowing a large number of possible variants and ensuring consistency between all variants is more difficult than in non-distributed systems. This is because highly distributed systems tend to lack a central control over different parts of the system to ensure consistency.

Challenges related to runtime variability. We identified three challenges related to variability at runtime.

First, we currently lack modeling techniques to describe variability at runtime. For example, there is still a debate in the community if we can use normal variability modeling techniques (such as feature models) to model runtime variability, or if we need specific modeling approaches [4]. Also, it is difficult to model all possible combinations of variants if many variation points and variants exist.

Second, there was a debate of what kind of (or how much) human involvement should be allowed when enabling runtime variability. The group agreed that there should be a balance between flexibility (i.e., lots of human involvement) and speed (i.e., automatic resolution of variability at runtime without the need of humans being involved in selecting variants). Automation has also been identified as a future research area by the group that discussed traceability of variability (see Section 2.3).

Third, the group discussed that there is a potentially very large number of variants, and a large number of possible combination of variants. This makes it difficult to evaluate and test architectures. To overcome this problem, we could apply testing or reviewing techniques from other disciplines, such as techniques used to test dynamic binding in service-oriented architectures [5].

Solutions to enable variability at runtime.

Potential solutions to facilitate runtime variability were identified. These are considered as areas for future research.

First, to overcome the problem of testing a large number of variants or combinations of variants, simulation techniques could be used, based on approaches for software product line testing [6] or search-based testing of non-functional system properties [7].

Second, to identify possible variation points and sources of variation, learning mechanisms (e.g., data mining approaches) to determine variation points based on previous and similar systems can be used.

Third, the group suggested the use of dynamic languages to describe, test and implement variability at runtime.

Finally, to express runtime variability, the group proposed the use of architectural viewpoints and views. In future work, viewpoints can be defined that combine variability concerns and runtime concerns (e.g., performance) to describe dynamic characteristics of the architecture and their impact on variability, or the impact of variability at runtime on quality attributes.

6.2 Reference Architectures versus Software Product Line Architectures

Reference architectures can be used to facilitate variability [8]. In software product line engineering, the term “reference architecture” is often used as a synonym for “software product line architecture”. However, there have been recent discussions about the differences between reference architectures and product line architectures [9]. Therefore, one discussion group explored the concept of reference architectures versus the concept of product line architectures.

The group agreed that reference architectures are mostly about standardization in a certain domain (application domains like automotive systems, or technology domains like middleware), rather than about standardization for a certain set of products (as it is the case for product line architectures). Participants from industry considered reference architectures as a key to software design as it facilitates standardization of interfaces and components. This facilitates system architects in selecting suppliers of system components that must comply with standards and interfaces prescribed in the reference architecture definition.

Furthermore, reference architectures may concern different aspects of systems (e.g., business processes or software components), or they can concern different parts of the same system (e.g., braking system of an automotive architecture). To express a system from the perspective of different stakeholders in a reference architecture, reference architecture descriptions could utilize the description of different viewpoints [3]. In contrast, product line architectures usually express systems in terms of features and refer to the whole system, rather than to different parts of the same system.

In terms of levels of abstraction, reference architectures tend to be more conceptual and abstract capturing the general system structure and behavior for an entire domain. Product line architectures are usually defined in a more fine-grained manner and describe more details about the variants in specific products. Reference architectures can be used in product line engineering by transforming the abstract reference architecture into a concrete product line architecture. To achieve this, reference architectures need to be refined by adding more details about variants and variation points in the various products in the product line.

This refinement can be accomplished by using principles from model-driven engineering:

1) Abstractions in the reference architecture can be instantiated into concrete components in the product line.
2) Variants for the different variation points can be specified and selected.
3) Quality attributes and other architectural drivers can be prioritized to serve the specific product line at hand.

A sub-topic that was discussed by the same discussion group concerned the evaluation of variability in the architecture. Since architecture evaluation methods use the architecture description as input, and particularly the architecture views, the group discussed that the outcome of the evaluation depends largely on the architecture views at hand. If for example an architecture framework such as the 4+1 views [10] is used, which has not explicit variability information, then the variability to be evaluated needs to be inferred. In cases where variability information exists within the architecture views, or even better in specialized variability views (e.g., [11]), the architecture evaluation with regard to variability can be performed more effectively. In any case, since variability is a horizontal concern in a system architecture, there needs to be good traceability between the architecture views: When evaluating the variability in one view (e.g. logical view) we also need to look into other views (e.g. deployment view) in order to examine system variability holistically (see also Section 2.3). This forces us to strive for effective traces between the different architecture views in general but also between specialized variability views (if they exist) and other views.

As with architecture evaluation in general [12], a key issue is to take trade-offs with other quality attributes into consideration. This is far from trivial, as sensitivity points and trade-off points need to be identified in all inter-related quality attributes. Furthermore, most architecture evaluation methods used in practice are scenario-based [13], which leads us towards considering defining variability scenarios in order to make variability more concrete and structured. Such scenarios could concern, for example binding variants to variation points and deriving combination of variants. In this sense each variability scenario would be evaluated against the main architecture design decisions that affect the scenario.

6.3 Traceability of Architectural Variability

The discussion group discussed perspectives on capturing variability and ensuring traceability of variability in software architectures. Traceability in architectural views is already complex without considering variability. However, traceability helps avoid consistency problems and allows stakeholders to navigate between variation points, variants, product versions and other architectural artifacts (e.g., design decisions). The group identified several cases of traceability with regard to variability in the architecture:

1) Traceability between separate variability views / models and other architectural model / views. Traceability needs to be ensured between different models, and different views, as well as within views.
2) Traceability between variability views. In addition to traceability between variability-specific models / views and other types of views / models, traceability needs to be ensured among different variability views.
3) Traceability between architectural views and other artifacts (e.g., code, tests).

The group found that that there are several approaches to communicate variability in architectural views.

1) Variability can be integrated into other software architecture artifacts or architectural models (e.g., components or connectors), similar as integrated variability modeling proposed by Pohl et al. [14]. In this approach, variation points, variants and relations are specified in existing architectural views and related models. This approach would be easy to implement, but allows only limited traceability because relationships between variability elements and other architecture artifacts are not explicit.
2) Variability can be described separately in an orthogonal variability description (orthogonal variability modeling [14]). This approach requires a separate view with variation points and variants as options to resolve variation points. It facilitates traceability, but adds complexity and requires significant effort as relationships between different models and views need to be identified and described.
3) A combination of both approaches can be applied. There may be a certain amount of variability (see also Section 2.1) which can be added to other architectural views. However, there was no agreement on how the combination could be put into practice. The amount of variability to add to a view would be determined by the fundamental needs of stakeholders. For example, if a middleware engineer is interested in all component candidates for an authentication mechanism, this variability point should be present in the view. Alternatively, a regular programmer might not be interested in a sequence diagram with components that s/he is not responsible for. Therefore, this judgment needs to be used when deciding on how much variability awareness to include into each view.

The group also proposed a research agenda, including research issues, directions and ideas related to the intersection of traceability and variability:

1) How can find a balance between separation of concerns and integration of concerns with regard to variability? This requires that we resolve the trade-off between the separation of concerns and integration of relevant variability information in single models. Here, we could merge metamodels of views with other metamodels to incorporate the needed variability information.
2) How can we decide about the amount of variability in the architecture? Whenever we implement traceability of architectural variability, we have to be careful to not add too much documentation overhead. Furthermore, cost-based reasoning can be applied to avoid spending effort on traceability but still ensuring benefits of implementing traceability mechanisms. Choosing a variability representation and the amount of variability in a software system is not trivial (see also Section 2.1). Too much variability can cause project failure. In general, variability in software should satisfy needs inside a domain or market, and add as little extra complexity as possible.
3) How can we facilitate automation in the context of variability to facilitate traceability? Automation would help implement traceability mechanisms. Here, automation can be applied to checking the consistency of models, the derivation of products from variability-intensive software architectures, or the creation of traceability links between different architectural models, variability models and even architecture views.
4) How can we integrate different variability management approaches in heterogeneous environments? It is still unclear how to integrate different variability management approaches when different organizations collaborate and bridge their boundaries. Thus, one question for research is to find ways to share designs that have been created following the orthogonal and integrated variability modeling paradigms. Another topic for future work is to find out how to achieve consensus when several organizations have variability requirements that need to be shared in one variability-intensive software architecture.
5) How can we ensure traceability through a single metamodel that is used to generate architectural views? The idea is to avoid spending effort for facilitating traceability by generating views from a single metamodel. However, there are reasonable doubts about the practicality of a single metamodel for modern software projects. It is very likely that those systems that have such a model simple enough to operate would not benefit from a first-class variability representation in the first place.

7. ACKNOWLEDGMENTS

We extend our thanks to all who have participated in the organization of the workshop, particularly submitters and presenters, workshop participants, the members of the program committee, and the WICSA organizers. This work has been partially supported by NWO SaS-LeG, contract no. 638.000.000.07N07.

8. REFERENCES

[1] M. Galster, D. Weyns, P. Avgeriou, M. Becker, Second International Workshop on Variability in Software Architecture, in: WICSA / ECSA 2012 Companion Volume, ACM, Helsinki, Finland, 2012, pp. 132-134.

[2] M. Galster, P. Avgeriou, D. Weyns, T. Männistö, Variability in Software Architecture: Current Practice and Challenges, ACM SIGSOFT Software Engineering Notes, 46 (2011) 30-32.

[3] ISO/IEC/IEEE, 42010 – Systems and Software Engineering - Architecture Description, Geneva, Switzerland, 2011.

[4] S. Hallsteinsen, S. Jiang, R. Sanders, Dynamic Software Product Lines in Service-oriented Computing, in: 3rd International Workshop on Dynamic Software Product Lines, San Francisco, CA, 2009, pp. 28-34.

[5] M. Palacios, J. Garcia-Fanjul, J. Tuya, Testing in Service-oriented Architectures with Dynamic Binding: A Mapping Study, Information and Software Technology, 53 (2011) 171-189.

[6] K. Pohl, A. Metzger, Software Product Line Testing, Communications of the ACM, 49 (2006) 78-81.

[7] W. Afzal, R. Torkar, R. Feldt, A Systematic Review of Search-based Testing for Non-functional System Properties, Information and Software Technology, 51 (2009) 957-976.

[8] M. Galster, P. Avgeriou, Empirically-grounded Reference Architectures: A Proposal, in: 7th ACM Sigsoft International Conference on the Quality of Software Architectures (QoSA), ACM, Boulder, CO, 2011, pp. 153-157.

[9] E.Y. Nakagawa, P.O. Antonino, M. Becker, Reference Architecture and Product Line Architecture: A Subtle but Critical Difference, in: 5th European Conference on Software Architecture, Springer Verlag, Essen, Germany, 2011, pp. 207-211.

[10] P. Kruchten, The 4+1 View Model of Architecture, IEEE Software, 12 (1995) 42-50.

[11] M. Galster, P. Avgeriou, A Variability Viewpoint for Enterprise Software Systems, in: Joint 10th Working IEEE/IFIP Conference on Software Architecture (WICSA) & 6th European Conference on Software Architecture (ECSA), IEEE Computer Society, Helsinki, Finland, 2012, pp. 267-271.

[12] P. Clements, R. Kazman, M. Klein, Evaluating Software Architectures: Methods and Case Studies, Addison-Wesley, Boston, MA, 2002.

[13] L. Dobrica, E. Niemela, A Survey on Software Architecture Analysis Methods, IEEE Transactions on Software Engineering, 28 (2002) 638-653.

[14] K. Pohl, G. Boeckle, F. van der Linden, Software Product Line Engineering - Foundations, Principles, and Techniques, Springer Verlag, Berlin / Heidelberg, 2005.