Variability in Software Architecture: Current Practice and Challenges

Matthias Galster
University of Groningen
The Netherlands
m.r.galster@rug.nl

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

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

Tomi Männistö
Aalto University
Finland
tomi.mannisto@aalto.fi

DOI: 10.1145/2020976.2020978
http://doi.acm.org/10.1145/2020976.2020978

Abstract

Variability in software-intensive systems is usually understood as the ability of a software artifact to be changed in order to fit different contexts, environments, or purposes. Software architecture on the other hand determines the structure of a software system, and is described in an architecture description. This description includes the major stakeholders of a software system and their concerns. Variability is reflected in and facilitated through the software architecture. The First International Workshop on Variability in Software Architecture (VARSA) was held jointly with WICSA 2011 in Boulder, Colorado. The goal of the workshop was to explore and advance the state-of-the art in variability in software architecture. It featured four research paper presentations, two invited talks, and three working groups that discussed specific topics. This report summarizes the themes of the workshop, presents the results of the working group discussions, and suggests topics for further research.

Introduction

Variability is the ability of a software artifact to be changed (e.g., configured, customized, extended, adapted) for a specific context, in a pre-planned manner [1]. This means, variability can be understood as "anticipated change". Variability a) helps manage commonalities and differences between systems, b) supports the development of different versions of software by allowing the implementation of variants within systems, c) facilitates planned reuse of software artifacts in multiple products, d) allows the delay of design decisions to the latest point that is economically feasible, and e) supports runtime adaptations of deployed systems. Reasons for variability include the deferral of design decisions, multiple deployment / maintenance scenarios, self-* systems, etc. Here, * can include things like healing, adapting, configuring, optimizing, protecting, etc. Mechanisms to accommodate variability include variability management tools for software product lines, configuration wizards and tools for commercial software, configuration interfaces of software components, infrastructures to support runtime composition of web services, etc. [2, 3].

Variability is primarily reflected in and facilitated through the software architecture. Furthermore, the software architecture is the centerpiece of software systems and acts as the reference point for many development activities (e.g., implementation, testing, maintenance), and many of today's software systems are built to accommodate variability. Thus, variability in software architecture should be well-understood and be treated as a first-class concern.

The software architecture community acknowledges that variability is a concern of different stakeholders, and in turn affects other concerns. Nevertheless, treating variability related to the architecture and all architecture aspects, as a cross-cutting concern, is currently not well understood. Therefore, the First International Workshop on Variability in Software Architecture (VARSA 2011) aimed at identifying critical challenges, and progressing the state-of-the-art on variability in software architecture. Around 35 participants were registered for the workshop. Further information about the workshop, its theme, motivation, etc. can be found in the WICSA proceedings [1] as well as on the workshop website [4].

Invited Talks

The workshop featured invited talks by Len Bass from the Software Engineering Institute (SEI), USA, and Juha Savolainen from Nokia Research, Finland.

Identifying and Evaluating Variation Points in Residential Demand Response Systems

Len Bass elaborated on the specifics of identifying variation points in smart grid systems, an example of ultra-large-scale systems. In particular, Len focused on demand response as used in smart grids to reduce peak loads during periods of high demand for electricity or in situations when grid reliability is at risk. In smart grids and demand response systems, scalability is the main concern. In contrast to other systems, smart grid systems are unprecedented systems. Consequently, different procedures for identifying variation points are needed.

In case of precedented systems, potential variation points are identified through choices that come from marketing or the examination of existing systems. Actual variation points are then decided based on the costs and benefits of placing the variation points in the product. On the other hand, in unprecedented systems, potential variation points can be identified through determining decision points when designing an architecture. Actual variation points can then be decided based on evaluating risks in possible architectures that would result from particular decisions.

According to the speaker, variation points can be treated as decision points. To identify variation points, he proposed the following three steps:

1. Identification of major decisions that must be made in order to build the desired systems. The result of this step is a list of decisions that are candidate variation points, and a set of alternatives for each decision.
2. Assessment of whether or not these candidate decisions are within the scope of the proposed product.
3. Evaluation of the alternatives and decisions with regard to whether they are alternatives to be included in the variation point. This means, variation points and variants are treated from the perspective of decisions and alternatives (for different contexts).

To analyze variation points, Len argued for six steps:

1. Identification of goals for the system.
2. Documentation of the most common anticipated use cases for the system.
3. Documentation of architectural alternatives.
4. Development of scenarios that describe challenges to the system from multiple quality attribute perspectives (reliability, performance, modifiability, usability, security, etc.).
5. Identification of potential risks.

6. Consolidation of risks into risk themes to allow for strategic planning.

These steps can be performed using the ATAM [5].
Furthermore, Len argued that variability in smart grids is currently only considered with regard to variability in functionality, and implications of variability in functionality on quality attributes.

Experiences in Managing Evolution of Large Product Lines

Juha Savolainen talked about his experience at Nokia with managing the evolution of large product lines.

With regard to quality attributes and variability, Juha argued that quality attributes are always attached to functions (i.e., there is nothing like a system-level quality attribute), rather than taking quality attributes as drivers for design decisions. Furthermore, Juha argued that variability is not a quality attribute by itself, but enables flexibility and productivity, which are quality attributes in product lines. In other words, variability in product lines helps achieve benefits of product lines. In order to decide whether an architecture is a good architecture with regard to flexibility and cost, there is always a trade-off between the cost of changes needed for the variant and the cost of creating the variant.

Furthermore, Juha argued for the need of a functional core in product lines. The functional core should not allow for any variation, is fundamental to the system, and cannot be designed iteratively. Furthermore, the functional core would not be equal to the set of all mandatory features as mandatory features could express a commitment to possible change (which is not allowed to the functional core). Instead, the functional core describes implicit commonalities (e.g., frameworks, development environment). Overall, the functional core represents a framework to make it easier to build new products as it includes previously made design decisions which help improve productivity. Enforcing a functional core for product lines means that managing variability is not only about variability itself, but also about the common core. For the architect, this could mean that decisions have to be made which are not compliant with current market needs: The common functional core should be applicable for designing new products, independent of the current market needs. An interesting open research problem is to define a set of criteria to determine whether new required functionality should become part of the common functional core or not. Currently this is usually done based on intuition and gut feeling.

Working Group Discussions

The workshop accepted four research papers for inclusion in the proceedings. The papers were presented in short 15 minutes presentations (for details about the accepted papers please refer to [1]). The invited talks as well as the paper presentations provided the starting points for the discussion in three working group sessions. The following topics were selected for further discussion:

- Variability and architecture description.
- Variability management as decision problem.
- Variability in model-driven engineering.

The topics were selected based on the interests of participants. The following sections elaborate on the results of the discussions in the working groups.

Variability and Architecture Description

We first brainstormed about how variability is currently described in software architecture documents. During the brainstorming it became clear that variability is often not explicitly described in software architectures which are designed outside the product line domain. In many cases, information about variability often exists implicitly in the mind of architects as tacit knowledge. Also, sometimes, plain API documentations are used to facilitate variability. Other frequently found ways of describing variability are simple layered or modularized component and connector models that contain constant elements and encapsulate parts of the system that may change (i.e., parts that are variable).

Some participants of the discussion stated that sometimes it is simply too complex to model all variability in the architecture. Also, new or better approaches for variability description in the software architecture might not exist or not be used because organizations would face new tooling issues. Furthermore, as stated by participants, the learning curve for new methods might not be considered as reasonable for industrial organizations.

On the other hand, the lack of variability description causes problems because architectures and models cannot be refactored. Also, no synchronization between different architecture models is possible. As stated by some participants from industry, metamodels for the derivation of views would be desired.

We then continued to explore ways for how variability could be described as explicit information in the software architecture documentation. Several options were identified:

- Informally, no standardized format: This means, variability is informally described as a concern of different stakeholders, but not as a first-class concern. Also, it means that variability information is not necessarily part of architecture models. Examples include the specification of API's in text documents as supplemental material for architecture descriptions, user manuals, variability descriptions in header files, or information about compiler settings to compile software for different platforms.
- Annotations: Existing architecture models can be annotated with variability information. For example, UML class diagrams can be enriched by adding stereotypes, or by attaching notes to elements in the class diagram to mark variation points or variants.
- Dedicated variability descriptions: Dedicated variability descriptions include specific variability models, variability views or the definition of variability viewpoints.

We concluded that a combination of all options might be applicable in certain situations.

Next, we discussed in what situation which description strategy might be applied. We identified the following main factors: First, the degree of variability (for example, in systems which have a low degree of variability, not much effort would be spent on describing variability in the architecture); second, the type of variability (variability in components might be easier to describe than variability in quality attributes or dynamic variability); third, organizational issues (e.g., culture, domain, life cycle or life time of product).

Topics for future research include the development of methods to ensure traceability and consistency between different levels of variability, and variability in different parts of the architecture. Furthermore, we found that there is a need for better tools to manage variability on an architectural level. This is particularly true for the use of new approaches in industry.

Variability Management as Decision Problem

We also discussed variability management as a decision problem. This discussion was inspired by the invited talk given by Len

Bass. In detail, we discussed the possibility of borrowing decision-making methods from other contexts to represent and reason about variability in the software architecture. In this context, we discussed how to adapt the notion of "decision" to describe variation points. We believe that this understanding could help when designing product families. However, we did not think that this would be equally helpful during product derivation.

Whenever an architect needs to make a decision, multiple alternatives exist. This means, if one does not select a single alternative (but just introduces a set of possible options), one is already introducing variability. We agreed that a decision topic is a topic for a decision that has to be made. In this sense, a variation point is a decision topic as architects or other stakeholder select the variant and associated variation points and variants. We believe that every variation point is a decision topic but not every decision topic is a variation point. Similar as with decisions and decision alternatives, alternatives would exist for variation points and variants (such as "enumerated", "multiple"). For the management of variation, we might adapt mechanisms for managing decisions.

Moreover, decisions would exist at different levels. There could be high-level decisions, architecture level decisions (full-fledged product architecture, a little bit of architecture, or a "quick and dirty" architecture), and finally the actual variant. We believe that in general high-level decisions do not result in variability. All the different levels and kinds of decisions are in some sense constraining variability from the later design phases, but as they are not treated (or managed) as variability, calling them variability would stretch the concept of variability too much, and thus not all decisions are "variability".

On the other hand, there is a temporal aspect. This includes the questions of when we need to make the decision, the order of decisions (something needs / is beneficial to be made before something else), as well as the temporal aspect of the relationships between decisions. We also discussed the point that single decisions are never made alone. This means that decisions are highly dependent and that there are trade-offs in the understanding of the consequences of defining variation points and choosing variants from the perspective of different views (e.g., user, server load, complexity of implementation, time to launch).

For dependencies between variants, we thought that Kruchten's work on architectural decision models and types of dependencies could be used (see [6]).

As a topic for future research we identified linking decision making and knowledge management. As we have already mentioned above, decisions can be treated as variability points. However, a clear understanding of the relationship between decisions and variability is still missing.

Variability in Model-driven Engineering

The group explored the notion of variability in the context of model-driven engineering (MDE) of software product lines. The discussion started from the observation that the transformation from platform-independent models (PIM) to platform-specific models (PSM) could be considered as the selection of a variant with respect to the platform. As a consequence, the choice for a model-driven engineering approach for software product lines may affect the way variability is modeled and modularized. Figure 1 shows an overview of how variability of a software product line may be modeled to support the transformation from PIM to PSM. The PIM is provided with a variability model that captures the platform independent variability (PIVM). The transformation specification is provided with a variability model that captures the variability with respect to the supported target platforms (PRVM). In general, dependencies will exist between the PIVM and PRVM. For example, the selection of a coordination mechanism (specified in the PIVM) may depend on the available communication technology of the selected platform (specified in the PRVM). These dependencies are captured in the platform variability dependency model (PVDM).

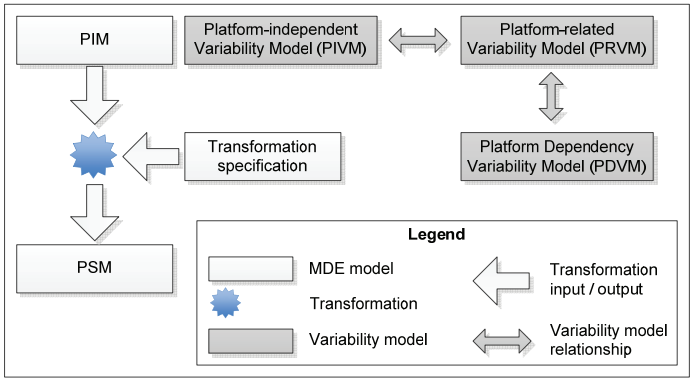

Figure 1. Variability in the context of MDE

In summary:

Variability model in MDE= PIVM + PRVM + PVDM

It is clear that Figure 1 shows a high-level view that should be refined for a concrete setting. For a SOA setting, the discussion participants believe that a mapping of the variability models should be rather straightforward since business logic and deployment are typically well separated. However, it is not clear how easy the proposed modularization of the variability model will be for specific domains. Therefore, one interesting opportunity for further research is to study how existing variability models (in particular models employed for product line engineering) support the proposed modularization of variability and how efficient such modularization will be for model-driven engineering of products.

Acknowledgments

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

References

[1] Galster, M., Avgeriou, P., Weyns, D., Männistö, T. (2011): First International Workshop on Variability in Software Architecture (VARSA 2011). In *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, IEEE Computer Society, pp. 280-281.

[2] Hilliard, R. (2010): On Representing Variation. In *Workshop on Variability in Product Line Architectures (VARI-ARCH)*, ACM, pp. 312-315.

[3] van Gurp, J., Bosch, J. (2003): Proceedings of the Software Variability Management Workshop.

[4] First International Workshop on Variability in Software Architecture, www.cs.rug.nl/~matthias/varsa2011.

[5] Clements, P., Kazman, R., Klein, M. (2002): Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley.

[6] Kruchten, P. (2004): An Ontology of Architectural Design Decisions in Software-intensive Systems. In *2nd Groningen Workshop on Software Variability*, pp. 54-61.