Variability for Qualities in Software Architecture

Azadeh Alebrahim
paluno - The Ruhr Institute for Software Technology
Working group Software Engineering
University of Duisburg-Essen, Germany
azadeh.alebrahim@uni-due.de

Stephan Faßbender
paluno - The Ruhr Institute for Software Technology
Working group Software Engineering
University of Duisburg-Essen, Germany
stephan.fassbender@uni-due.de

Martin Filipczyk
paluno - The Ruhr Institute for Software Technology
Working group Specification of Software Systems
University of Duisburg-Essen, Germany
martin.filipczyk@paluno.uni-due.de

Michael Goedicke
paluno - The Ruhr Institute for Software Technology
Working group Specification of Software Systems
University of Duisburg-Essen, Germany
michael.goedicke@paluno.uni-due.de

Maritta Heisel
paluno - The Ruhr Institute for Software Technology
Working group Software Engineering
University of Duisburg-Essen, Germany
maritta.heisel@uni-due.de

Uwe Zdun
Research Group Software Architecture
University of Vienna
uwe.zdun@univie.ac.at

DOI: 10.1145/2853073.2853094

ABSTRACT

http://doi.acm.org/10.1145/2853073.2853094

Variability is a key factor of most systems. While there are many works covering variability in functionality, there is a research gap regarding variability in software qualities. There is an obvious imbalance between the importance of variability in the context of quality attributes, and the intensity of research in this area. To improve this situation, the First International Workshop on VAriability for QUalIties in SofTware Architecture (VAQUITA) was held jointly with ECSA 2015 in Cavtat/Dubrovnik, Croatia as a one-day workshop. The goal of VAQUITA was to investigate and stimulate the discourse about the matter of variability, qualities, and software architectures. The workshop featured three research paper presentations, one keynote talk, and two working group discussions. In this workshop report, we summarize the keynote talk and the presented papers. Additionally, we present the results of the working group discussions.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

Keywords

Software architecture, quality attributes, variability

1. INTRODUCTION

"Variability is a "key factor of most, if not all, systems" [6] and therefore a relevant concern of those systems" [5]. "Variability is the ability of a software system to be adapted for different contexts. Variability affects functionality as well as quality attributes of software systems. Even though variability is primarily studied in the software product line (SPL) domain, variability is a concern not only in the context of product lines but of many systems, including service-based systems" [7].

Variability is a frequently used key concern in Software Product Line Engineering (SPLE). Variability in SPLE represents the set of properties which vary in those products. Variability caused by introducing quality attributes has a significant impact on the architecture of an SPL as reported in a recent survey [8]. For SPLE, most works that cover variability only take functionalities into account. Additionally, many works that analyze variability in quality attributes only focus on qualities that are related to product lines such as flexibility, while software quality attributes are often neglected [4].

Besides software product lines, many software systems are designed to support variability, either at design time (for example, as product lines or product families) or at runtime, especially for self-adapting or service-based systems. Variability itself comes in two dimensions: functional variability and variability in software qualities. While there are many works covering variability in functionality, there is a research gap regarding variability in software qualities [5]. This gap is in contrast to the importance of variability in software engineering.

There is an obvious imbalance between the importance of variability in the context of quality attributes and the intensity of research in this area. Therefore, the First International Workshop on Variability for Qualities in Software Architecture (VAQUITA 2015) aimed at investigating and stimulating the discourse about the matter of variability, qualities, and software architectures. We focus on software architecture since it directly influences the fulfillment of a software system's quality attributes. Therefore, it requires special attention. In addition, the goal of the workshop was to bring together researchers and practitioners to share ideas and experiences, analyze research trends and upcoming research challenges, discuss open problems, and propose promising solutions with a particular focus on handling variability in software architecture with respect to quality attributes. Further information about the workshop, its theme, motivation, and the organizing and the program committees can be found in the workshop proceedings [1] as well as on the workshop website[1].

2. KEYNOTE TALK

The workshop featured a keynote talk titled "*Architecting for Variability in Quality Attributes of Software Systems*" by Matthias

[1]http://vaquita-workshop.org/

Galster from University of Canterbury, Christchurch, New Zealand.

Matthias Galster divided his talk into four parts. In the first part *fundamentals*, he talked about quality attributes, variability, and software architecture. According to Matthias, variability occurs not only in product lines, but also in other systems such as self-adaptive, configurable or customizable single systems, open platforms, or systems that support the dynamic composition of services. In addition to variability in functionality, it also occurs in quality attributes and quality attribute requirements. In the second part of the talk, Matthias introduced a scenario-based taxonomy for variability in quality attributes. He presented five scenarios, in which Scenarios 1,2 and 5 express "intentional" variability whereas the other scenarios express "unintentional" variability:

Scenario 1 Variability in quality attributes due to variations in the user and user needs, such as different market segments or customer profiles.

Scenario 2 Variability in quality attributes in different phases of the software development process. For example, during different testing phases or integration phases, privacy may not be a concern, whereas for final delivery of a multi-tenant SaaS product, privacy is a concern.

Scenario 3 Variability in quality attributes due to the impact of variability in functionality. For example, complex data visualization may affect performance.

Scenario 4 Variability in quality attributes due to the impact of variability in other quality attributes. For example, a higher response rate (performance) may reduce security.

Scenario 5 Variability in quality attributes due to variation in hardware resources. Examples for this scenario are variation in network capacity, variation in product hardware chipset, etc.

In the third part of his talk, Matthias reviewed the state of research regarding variability in quality attributes. He discussed five studies. Some of the main findings of the related work review are:

- Variability in quality attributes has not been subject of extensive research. When dealing with variability in quality attributes, performance and availability are addressed most whereas security and safety have not been investigated extensively.
- In service-based systems, studies use mostly laboratory settings rather than real-life case studies.
- In service-based systems, variability is mostly accommodated in the activities *architecture design* and *implementation and integration*.

According to the speaker, future research in the context of variability in quality attributes (fourth part of the talk) can be performed in two directions "*conducting exploratory and descriptive studies to better understand variability in quality attributes*" as well as "*devising approaches for describing, analyzing, and implementing variability in quality attributes*".

3. PAPER PRESENTATIONS

We accepted three research papers for inclusion in the proceedings. The papers were presented in short 15 minutes presentations. Each paper presentation was followed by a discussion which has been opened up by a pre-assigned discussant, who gave a critical review of the paper.

Models for Self-Adaptive Systems

The first paper was presented by Amir Molzam Sharifloo. It focused on the roles of models and the relationships among them in self-adaptive systems. It classified the types of models often required for self-adaptation into environment, system, and requirements. The environment is defined as everything on which the system does not have any control but can impact on system functionality. Defining the scope has been identified as the main challenge in modeling the environment. Modeling and specifying requirements has always been a challenge in software development. Although much research has been done on this topic in the recent years, more work is still required. The system architecture as the central concept for developing self-adaptive systems has to contain variation points, knowledge about design decisions, and rationales behind the design decisions which are vital for adaptation planning. Furthermore, Amir pointed out the following main challenges of using run-time models by self-adaptive systems: model integrity, variability modeling, uncertainty, tacit architectural knowledge, and run-time verification. This paper can be found in [10].

Automatic selection and composition of model transformations alternatives using evolutionary algorithms

The second paper [9], presented by Smail Rahmoun, focused on making trade-offs between non-functional properties when presented various design alternatives for software architectures. Therefore, Rahmoun et al. formalize these design alternatives with model transformations, use evolutionary algorithms to create model transformation alternatives and finally identify the transformation that represents the best trade-off between the non-functional properties at hand. From a software architecture given as AADL model, an initial population of model transformation alternatives is created. After the non-functional properties of all alternatives are evaluated, a new generation of offspring is created using the evolutionary operation mutation and crossover. Finally, the output of the method presented in [9] is a set of nearly optimal AADL models. The authors demonstrate their approach in a use case with the goal of finding the optimal deployment of interconnected software components on hardware components.

Evolving a Software Products Line for E-commerce Systems: A Case Study

The third paper [3], presented by Leonardo Montecchi, deals with evolution of software product lines (SPL). Azzolini et al. present a case study for ACFAM, an aspect-oriented and component-based methodology for implementing software product lines. In their case study, they compare the ACFAM approach with other types of SPL implementations to prove its applicability to real-world domains and its higher effectiveness compared to other implementations. Therefore, Azzolini et al. evaluate the ACFAM approach using a real-world scenario. They extract a feature set from four existing e-commerce systems, identify commonalities and variabilities, and set up a feature model using the ACFAM methodology. Additionally, they design an evolution scenario containing four releases which are implemented using the different SPL approaches. The authors' findings indicate that ACFAM is indeed applicable for real-world scenarios and is more effective for product line archi-

tectures in e-commerce domains since it required less code changes and provided the most stable architecture.

4. WORKING GROUP DISCUSSIONS

The two following topics were selected for discussing in two parallel working groups:

- Complexity of the Design Space
- Quality Attributes and Variability

We selected these two topics according to the interests of the workshop participants. In the following, we elaborate on the results of the discussions in the working group sessions.

Complexity of the Design Space

The discussions in this working group targeted the questions why variability makes our systems more complex and what we can do about it.

Sources of Complexity A source of complexity in variability-aware systems is the set of relationships and dependencies among various quality attributes as well as the interconnection between quality attributes and functionality. Sometimes, functionality has to be dropped to fulfill a particular quality attribute.

Reduce Complexity One key agreement of the discussion was that variability should only be considered if the software engineer is absolutely confident that it will be needed during the lifetime of a system. Since variability introduces complexity, the rule of thumb should be *Don't implement variability you do not need*. Then again, if there is already built-in variability support regarding a particular aspect and observe that you will never need it you may consider removing it to increase the overall maintainability of the system.

Unforeseeable Variability There are systems that experience variability at runtime that has not been foreseen when designing the system, or cannot be controlled, or both. For this case, three strategies have been proposed: First, focus the implementation on the aspects that are known beforehand. If the software engineer starts guessing what might happen, the complexity of the system is unnecessarily increased (cf. bullet point *Reduced Complexity*). Second, the software engineer may design the system with the mindset that it can and will fail at runtime in case of unforeseen variability – the knowledge about variability acquired in this case can be seen as a new aspect that is known beforehand for another release of the system (in terms of the first strategy). Of course, this strategy is not always applicable, which may be the case for safety-critical system, e.g. medical or avionic systems. A third strategy comprises the data-driven definition of rules that triggers a reaction for observed values. The premise for this strategy is the impossibility of defining a reaction to every possible chain of events or events that may occur simultaneously.

Variability in Middleware Regarding the technical implementation of variability, most modern middlewares are Interceptor-based, i.e. they heavily rely on the Interceptor pattern (cf. [2]). On one hand, from a structural viewpoint, the architecture is rather static and in itself is not variable. On the other hand, the Interceptor instances itself support variability since the programmer can implement arbitrary code. There is obviously a difference between variable architectures and architectures that support variability.

Future Research Obviously, variability in software architectures is a complex challenge, especially considering quality attributes. For the future, we can envision various areas for research regarding these challenges. Architectural knowledge captured in architectural patterns and tactics seems to be a good source for solutions to said challenges. Since making general assumptions about all software systems is difficult, one idea is to create context-specific pattern and tactic catalogs. Feature interaction, especially feature modeling or goal modeling between quality attributes could be a promising area of research. Finally, the establishment of variability viewpoints on software architecture should be investigated in the future.

Quality Attributes and Variability

During the workshop, we identified that a common understanding regarding the term "variability" does not exist. Hence, we first brainstormed about how variability is currently defined. During the brainstorming, it became clear that depending on the context or application domain in which the term variability is used such as software product line or self-adaptation, there might existing different understandings of the term variability. However, within a particular context or application domain, the term variability is quite clearly defined and understood.

Next, we discussed about variability and product lines considering quality attributes on the architecture level. We identified that it might be too complex to come up with a (reference) architecture with sufficient commonalities when applying architectural patterns for addressing different quality requirements. The problem identified in this context is that on the one hand quality requirements are mostly conflicting and on the other hand architectural patterns contributing to the satisfaction of those requirements come always with some benefits for one type of quality requirements and reliabilities for the other type of quality requirements. This makes the decision about selecting the architectural patterns for a (reference) architecture complex. However, the workshop participants discussed four potential solutions to deal with this dilemma:

Early design decisions Early at the architectural level, it should be decided on the satisfaction of only one type of quality requirements which are not conflicting. This early design decision allows the selection of an architectural patterns contributing to the satisfaction of the selected quality requirement. This idea is, however, in contrast to the variability and product line concepts.

Patterns in application engineering Architectural patterns should not be applied in the domain engineering but in the application engineering. This solution might need some refactoring work as it might be probably too late to apply an architectural pattern at this level.

No common architecture There might not exist a (reference) architecture which accommodates the commonalities of all conflicting quality requirements. The reason is that quality requirements are considered architectural drivers. Different conflicting quality requirements might lead to completely different software architectures using different architectural patterns so that no common architecture can exist.

Patterns and tactics An architectural pattern positively contributing to one type of quality requirements and negatively contributing to the other types of quality requirements should be applied in the domain engineering. In the application engineering, tactics should be used to compensate the negative effect of the architectural pattern on the one type of quality requirements. Also this solution provides no optimal solution to the problem at hand.

5. ACKNOWLEDGEMENTS

As the workshop organizers, we would like to thank the speakers for presenting their work on the topics of VAQUITA as well as all the participants for fruitful discussions. Additionally, we thank the members of the program committee for reviewing the submissions to this workshop. We would like to express a special thanks to our keynote speaker Matthias Galster. We appreciate the effort he, Magnus Standar, and Leonardo Montecchi put into reviewing this workshop report. Finally, we thank Matthias Galster in his role as workshops chair who assisted us in the preparations of the VAQUITA workshop. This work was partially supported by the German Research Foundation (DFG) under grant numbers HE3322/4-2 and GO774/5-2.

6. REFERENCES

[1] A. Alebrahim, S. Faßbender, M. Filipczyk, M. Goedicke, M. Heisel, and U. Zdun. 1st Workshop on VAriability for QUalIties in SofTware Architecture (VAQUITA): Workshop Introduction. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 22:1–22:2. ACM, 2015.

[2] P. Avgeriou and U. Zdun. Architectural Patterns Revisited – A Pattern Language. In *In 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee*, pages 1–39, 2005.

[3] R. P. Azzolini, C. M. F. Rubira, L. P. Tizzei, F. N. Gaia, and L. Montecchi. Evolving a Software Products Line for E-commerce Systems: A Case Study. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 26:1–26:7. ACM, 2015.

[4] L. Etxeberria and G. Sagardui. Variability driven quality evaluation in software product lines. In *SPLC*, pages 243–252, Sept 2008.

[5] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou. Variability in Software Systems – A Systematic Literature Review. *TSE*, 40(3):282–306, 2014.

[6] R. Hilliard. On Representing Variation. In *ECSA: Companion Volume*, pages 312–315. ACM, 2010.

[7] S. Mahdavi-Hezavehi, M. Galster, and P. Avgeriou. Variability in quality attributes of service-based software systems: A systematic literature review. *IFSOF*, 55(2):320 – 343, 2013. Special Section: Component-Based Software Engineering (CBSE).

[8] A. Metzger and K. Pohl. Software Product Line Engineering and Variability Management: Achievements and Challenges. In *FOSE*, pages 70–84. ACM, 2014.

[9] S. Rahmoun, E. Borde, and L. Pautet. Automatic Selection and Composition of Model Transformations Alternatives Using Evolutionary Algorithms. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 25:1–25:7. ACM, 2015.

[10] A. M. Sharifloo. Models for Self-Adaptive Systems. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 24:1–24:5. ACM, 2015.