Twin Peaks goes Agile

Matthias Galster
University of Canterbury
Christchurch, New Zealand
mgalster@ieee.org

Mehdi Mirakhorli
Rochester Institute of Technology
Rochester, NY, USA
mehdi@se.rit.edu

Anne Koziolek
Karlsruhe Institute of Technology
Karlsruhe, Germany
koziolek@kit.edu

DOI: 10.1145/2815021.2815038
http://doi.acm.org/10.1145/2815021.281503

ABSTRACT
Activities related to software requirements engineering and software architecture significantly contribute to the success of software development projects. In software engineering practice, requirements and architecture affect each other and should not be treated in isolation. However, from a research and conceptual perspective, the dependencies between requirements and architecture are usually investigated by focusing on either requirements engineering or software architecture. Therefore, the Fifth International Workshop on the Twin Peaks of Requirements and Architecture explored the relationship between requirements and software architecture in the broader context of software engineering in general. Based on the outcomes of previous editions of the workshop, this edition aimed at exploring the relationship between requirements and architecture in the context of agile software development. In general, the discussions at the workshop centered on proactive versus reactive software architecture engineering. We found that proactive architecture engineering provides benefits in certain contexts and domains. However, in order to succeed in competitive and fast-moving markets, reactive software architecture engineering may be more suitable, even though reactive architecture engineering could be difficult to in practice since it may require a different mindset from requirements engineers and architects.

Keywords
Requirements engineering, software architecture, Twin Peaks model.

1. INTRODUCTION
The Twin Peaks model proposed by Bashar Nuseibeh [1] acknowledges that the disciplines of requirements engineering and software architecture and the work products of these disciplines (i.e., software requirements and architectural artifacts) cannot be treated in isolation. Requirements are constrained by what is technically and economically feasible. At the same time, feedback from the architecture requires renegotiating architecturally significant requirements with stakeholders. Therefore, the Twin Peaks model advocates incremental development based on intertwining software requirements and architecture. However, from a research and conceptual perspective, the dependencies between requirements and architecture are usually investigated by focusing on either requirements engineering or software architecture. Therefore, the Fifth International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks@ICSE 2015) was dedicated to exploring the challenges introduced by the relationship between requirements and architecture. TwinPeaks@ICSE 2015 was held in conjunction with the 37th International Conference on Software Engineering (ICSE 2015) in Florence, Italy.

The goal of this workshop was to bring researchers, practitioners and educators from the RE and SA communities together

- to explore the state-of-the-art in software engineering, software architecture and requirements research, practice and education,
- to discuss methods, technologies and tools to "connect" the artifacts of requirements and architecture, and to better relate the disciplines of requirements engineering and software architecture, and
- to identify emerging trends related to the transition and the relationship between requirements engineering and software architecture.

Based on the results of previous TwinPeaks workshops, the theme for 2015 was "Twin Peaks goes Agile". We considered exploring lightweight techniques for integrating requirements and architectural thinking into the agile process, when to engage in upfront requirements analysis and architectural design, and techniques for incremental requirements architecture and product delivery.

Around 25 participants from industry and academia were registered for the workshop. One of the participants was Bashar Nuseibeh, the proposer of the original Twin Peaks model. The workshop was a follow-up event of TwinPeaks@RE 2012, TwinPeaks@ICSE 2013 [2], TwinPeaks@RE 2013, and TwinPeaks@ICSE 2014 [3].

2. PRESENTATIONS
Based on a peer review process, the workshop selected five papers for presentation and inclusion in the proceedings. Submitted papers were authored by academics and software engineering practitioners from all over the world. Papers were presented in 15 to 20 minute presentations. Papers presented at the workshop are available in the IEEE Xplore Digital Library.

Furthermore, the workshop featured an invited talk delivered by Anne Koziolek from the Karlsruhe Institute of Technology (KIT), Germany. Anne talked about design assistants for supporting the Twin Peaks and therefore contributed to our discussions on methods, technologies and tools to "connect" requirements engineering and software architecture. Anne started off by describing the use of models to predict the properties of a system in other engineering disciplines, such as computer modeling of building structures in civil engineering or power flow models for power systems planning. However, she argued that designing software is different, so the comparison with other engineering disciplines is somewhat limited: Building software (in the sense of compiling source code that is based on a software design) is almost free, in contrast to building physical constructs in civil engineering where the building process takes most of the resources. Also, with software, the executable system can directly and for free be generated from the detailed specification (i.e., the code). Still, predicting properties of software systems before writing all of its code would be desirable to detect bad design decisions early. To make modeling cost efficient, an idea is to maximize synergies between different modeling efforts to reduce the effort required to design each individual model. As possible model synergies, Anne mentioned the reuse of knowledge (about patterns, styles, or technologies), the prediction of properties (such as performance, reliability, or safety), documentation (such as the rationale for decisions to support evolution and maintainability of the architecture) and traceability (to support the evolution of requirements). Based on this notion, Anne presented the idea of a software architecture design assistant [4]. The term "architecture design assistant" can be used for any tool that takes as input some form of requirements for the system to be architected (and potentially even an initial design) and interacts with the architect to help her focus on areas of the design that could be improved to satisfy

requirements or optimized to improve quality or reduce costs. The idea of architecture design assistants is to assist the architect in a partially automated way and to combine different analyses to maximize synergies. From the perspective of requirements engineering and the Twin Peaks model, a software architecture design assistant is useful as it provides an explicit model of the software architecture and therefore provides immediate understanding of the consequences of new or changed requirements, also in agile projects. In the discussion following Anne's talk, the need to synchronize these models with "the real world", i.e., with the system implementation, was stressed. If the models are disconnected from the underlying system, the design assistants will produce inaccurate analysis results or even wrong suggestions. Additionally, as models do not necessarily help reduce design efforts, their creation must be cheap to be cost-effective. Here, the opportunity to create libraries of reusable model parts for different domains was mentioned. A participant envisioned that a future design assistant tool will retrieve such model parts from the internet just as today's operating systems retrieve drivers needed for new devices.

The workshop featured an afternoon keynote delivered by Jan Bosch, professor at Chalmers University of Technology, Sweden. Jan is also the director of the Software Center at the Chalmers University of Technology. The Software Center (http://www.software-center.se/) is a software research and innovation center with Ericsson, AB Volvo, Volvo Cars, Saab EDS, Axis Communications, Jeppesen and Grundfos as industry partners. The goal of the center is to increase the productivity of software development by a factor of 10 in the next ten years by addressing challenges around continuous delivery, continuous architecture, metrics, and customer data and ecosystem-driven development. Jan talked about architecting to ensure requirements relevance. Jan argued that up to two thirds of features in software systems are hardly ever used or not even used at all. This problem seems to occur across the software industry and therefore represents a tremendous waste of research and development efforts in the software industry. On the other hand, product management, engineers, developers and many others work hard at interacting with customers, building business cases and prioritizing requirements to deliver value to customers. Therefore, Jan argued, a fundamentally different approach to deciding what to build is required: requirements should be treated as hypothesis throughout the development process and constant feedback from users and systems in the field should be collected to dynamically reprioritize and change requirements. This requires architectural support beyond the current state-of-practice. Abilities to enable continuous deployment, separate testing and collecting data about the use of features need to be an integral part of the architecture. Jan presented a brief overview of the research and industry collaboration to address this challenge. More details about Jan's keynote can be found in a paper that complements the keynote [5].

We also invited workshop participants to present thoughts, ideas, questions, etc. in one minute statement presentations. Some participants used slides or a flipchart to illustrate their ideas. These statements covered a topic, question, idea or thought that participants were passionate about and interested in discussing with other workshop participants, or ideas that emerged throughout the workshop.

3. DISCUSSION

3.1 Initial Discussions

The goal of the discussions was to extend the findings from previous editions of the TwinPeaks workshops. Some initial questions that emerged during the workshop were the following:

- From a requirements engineering perspective, what constitutes software architecture design? In the architecture community, there seems to be an agreement of what software architecture and architecture design is (this includes technical artifacts but also things like architectural knowledge, the actual design, design decisions, design rationales, etc.). However, in the requirements engineering community, the understanding of the notion of software architecture is not as clear. Some participants argued that "software architecture design = choosing frameworks and mapping requirements onto elements of these frameworks". This notion is rather technical and, from a software architecture community point of view, does not really capture the complexity and challenges related to software architecting. Related questions were about the role of creativity when creating architectural designs (i.e., architecting is more than "mapping"), about how willing (and capable) software architects are to question and challenge requirements, and about the relationship between architecture, detailed design and code (as an implementation of the architecture and the design), and the required level of detail of architectural design in general.
- What is still missing to better understand the relationship of requirements and architecture? Participants argued that concrete use cases of transitions between requirements and architecture (including generic characteristics of these use cases) would help. These could then be used as templates for the transition between requirements and architecture. Furthermore, empirical data (including generic characteristics to describe the applicability of that data) and effort estimates about relating requirements and architecture would help in practice. Finally, guidelines for a transition or mapping and advice for training and education (i.e., integrated requirements engineering and software architecture courses) would help further narrowing the "gap" between requirements and architecture. These could be supported by models and tools.

These questions are rather fundamental. Given that this was the fifth edition of the workshop it indicates that a conceptual gap still seems to exist between the requirements engineering and software architecture communities.

3.2 Reactive Architecture Engineering

The remainder of the discussions at the workshop centered on the topic of *reactive architecture engineering*. Reactive architecture engineering means that only those customer requirements are implemented and designed in the architecture that provide immediate value to customers. Therefore, only those architectural design decisions are made that lead to immediate customer value. Architects would not try to anticipate future needs since this could be a waste of efforts and resources. Any future needs are implemented when requested by the customer; thus, many design decisions are delayed to the latest point possible. Also, reactive architecture engineering means that data about the actual use of features is collected. If features are not used, they may be removed in future releases if this makes the implementation of more important features easier.

The idea of reactive architecture engineering is partially in line with agile software engineering where the focus is on delivering value to customers. Therefore, reactive architecture engineering could be considered as one concrete practice applied in agile software engineering.

The idea of reactive architecture engineering would imply that we push mechanisms to quality attributes to the infrastructure (e.g., scalability is taken care of by a cloud-based infrastructure). Therefore, compliance with and consistency between quality attribute requirements could be delegated to the infrastructure, a benefit argued for in the context of cloud computing and service-oriented architecture. This would not only reduce complexity and simplify design decisions, but also support the need for speed to succeed in competitive markets, and increase flexibility to react to changing market and customer conditions. Some additional observations with regard to proactive architecture design are listed in Table 1.

Table 1. Comparison of proactive and reactive architecture engineering in the context of Twin Peaks

Proactive	Reactive
May lead to "overarchitecting" a solution, i.e., wasting effort that may not be required or that may not lead to customer value	May require a lot of refactoring to accommodate new needs, i.e., more effort may need to be spent later on
May be appropriate in less mature domains because requirements engineers and architects need to properly explore, understand and describe the problem space; also, in mature domains, architects may still need lots of experience to properly describe a fully-fledged architecture	May be more appropriate in mature domains where lots of "tacit" knowledge exists and architects have lots of experience (even more than for proactive architecture engineering)
May be more appropriate if the underlying technology is new, not yet well-understood or if a new technology is to be developed as part of the project	May be more appropriate if frameworks are reused and the underlying infrastructure or technology architecture is fixed (such as choosing a Java EE framework), i.e., if a more straight-forward "mapping" of requirements onto framework elements is possible
May give customers the perception that engineers do a thorough job; however, it may also lead to false confidence in the quality of a solution	May imply that the architect is doing her job poorly (i.e., if the architect did do her job properly, why do we need to change the architecture later on)
May result in overanalyzing solutions and anticipating what customers "may" want or need	May result in software architecture debt
Requires identification of core features and core attributes	Requires ignorance towards things that may be important but for which no evidence for their importance exists
Architecture methods exist (e.g., Attribute-driven design)	No systematic methods exist; mostly informal, based on intuition

The question of whether reactive architecture engineering is successful most likely depends on the "domain", i.e. application domains, technological domains, project types, etc. Given the comparison in Table 1, perhaps the Twin Peaks model could be better described better as Twin Peaks with "boulders", i.e., core requirements exist in one peak, and core architecture design decisions are described in the other peak, with "boulders" representing whatever additional work is done towards proactive architecture engineering (e.g., more, and more detailed requirements, more, and more detailed architecture design decisions) or whatever existing solutions can be reused in the form of frameworks.

4. CONCLUSIONS

Bashar Nuseibeh proposed the Twin Peaks model as a conceptual approach to reason about the relationship between requirements engineering and software architecture, and to highlight the role of change management throughout software development projects. In the workshop keynote, Jan Bosch emphasized the need for requirements and architecture co-evolution and the "need for speed" and feedback about the use of features in software systems to ensure that we architect systems with requirement relevance. Reactive architecture engineering may be the next step to combine the intertwinement of requirements and architecture while at the same time focusing on delivering customer value at high speed.

5. ACKNOWLEDGMENTS

We would like to thank all who have contributed to organizing and running the workshop. We are particularly grateful to submitters of papers, presenters, and workshop participants. Furthermore, we would like to thank the members of the TwinPeaks program committee and the ICSE organizers for supporting our workshop.

6. REFERENCES

[1] Nuseibeh, B. 2001. Weaving Together Requirements and Architecture. *IEEE Software* 34, 115-117.

[2] Galster, M., Mirakhorli, M., Cleland-Huang, J. Burge, J., Franch, X., Roshandel, R., Avgeriou, P. 2013. Views on Software Engineering from the Twin Peaks of Requirements and Architecture. *ACM SIGSOFT Software Engineering Notes* 38, 40-42.

[3] Galster, M., Mirakhorli, M., Cleland-Huang, J., Franch, X., Burge, J., Roshandel, R., Avgeriou, P. 2014. Towards Bridging the Twin Peaks of Requirements and Architecture. *ACM SIGSOFT Software Engineering Notes* 39, 30-31.

[4] Champagne, R., Koziolek, A. 2015. *Proceedings of the 1st International Workshop on Future of Software Architecture Design Assistants* (Montreal, Canada), ACM, New York, NY, USA. 1-24.

[5] Bosch, J. 2015. Architecting to Ensure Requirement Relevance. In *Proceedings of the Fifth International Workshop on the Twin Peaks of Requirements and Architecture* (Florence, Italy), IEEE Computer Society, 1-2.