Research Preview: Prioritizing Quality Requirements based on Software Architecture Evaluation Feedback

Anne Koziolek

Department of Informatics, University of Zurich, Switzerland
koziolek@ifi.uzh.ch

Abstract. [**Context and motivation**] Quality requirements are a main driver for architectural decisions of software systems. Although the need for iterative handling of requirements and architecture has been identified, current architecture design processes do not provide systematic, quantitative feedback for the prioritization and cost/benefit considerations for quality requirements. [**Question/problem**] Thus, in practice stakeholders still often state and prioritize quality requirements before knowing the software architecture, i.e. without knowledge about the quality dependencies, conflicts, incurred costs, and technical feasibility. However, as quality properties usually are cross-cutting architecture concerns, estimating the effects of design decisions is difficult. Thus, stakeholders cannot reliably know the appropriate required level of quality. [**Principal ideas/results**] In this research proposal, we suggest an approach to generate feedback from quantitative architecture evaluation to requirements engineering, in particular to requirements prioritization. We propose to use automated design space exploration techniques to generate information about available trade-offs. Final quality requirement prioritization is deferred until first feedback from architecture evaluation is available. [**Contribution**] In this paper, we present the process model of our approach enabling feedback to requirement prioritization and describe application scenarios and an example.

1 Introduction

Quality attributes such as performance, reliability, and maintainability, are crucial for the success of any software system. The software architecture largely influences the quality properties a software system will exhibit.

However, while quality requirements are defined in many companies mainly upfront, they are not systematically incorporated during development and thus are often dismissed later [2, 3]. In particular, interdependencies and trade-offs among quality requirements often remain unclear. Major difficulties complicate quality requirements prioritization tasks: First, quality attributes are often pervasive, so that their effect and costs are difficult to estimate in advance [2, pp. 3,9]. Second, for many types of quality requirements, a value on a continuous

scale, such as a response time of 5 seconds, needs to be defined. Choosing the right required value (i.e. the required *level* of quality, which is a subtask of requirements prioritization) is difficult for managers [3, p. 74].

Although the need for iterative handling of requirements and architecture has been identified decades ago, and several processes have been proposed [13, 14], no approaches provide *systematic and quantitative feedback* from software architecture design to support quality requirement prioritization.

Quantitative architecture evaluation approaches allow to predict quality properties (such as performance [10] and reliability [9]) based on models of the software architecture and underlying theories (such as queueing networks or Markov chains). They improve design decisions with respect to quality attributes and help to understand the incurred costs. However, these approaches assume fixed quality requirements and thus try to help the software architect to achieve these requirements, thus not reflecting the iterative nature of the development process.

As the contribution of this paper, we propose a new approach to prioritize quality requirements, relying on feedback from architecture evaluation and automated design space exploration. The approach requires identification of relevant quality attributes upfront but defers the decision for required quality levels. Only after initial architecture evaluation and design space exploration, the trade-off between quality attributes and the costs for achieving quality levels can be reliably estimated. To validate our research idea, we will (1) extend the existing design space exploration tool PerOpteryx [11] to explicitly support quality requirements prioritization and (2) evaluate its benefits in empirical studies, which include business reporting and industrial automation systems. The expected results of our approach are (1) better informed quality level definition, (2) guidance in quality requirement prioritization, and, as a result, (3) higher trust in quality requirements during the development process. Ultimately, our approach shall enable iterative handling of quality requirements and architecture.

The remainder of this paper is organized as follows. In Sec. 2, we discuss the current state and related approaches in more detail. Then, Sec. 3 describes our idea how to bring quality requirements and software architecture closer together and enable feedback. Finally, Sec. 4 concludes.

2 Related Work

The need for iterative handling of requirements and architecture has been identified decades ago [5]. The Twin Peaks model [13] suggests to concurrently develop requirements specification and architecture by using insight from one activity in the other. Woods and Rozanski [14] describe how insight from software architecture design can frame and inspire requirements specification. However, while both methods describe a mindset for software architects, they do not provide concrete methods and tool support to combine the two worlds.

2.1 Quality Requirements in Software Architecture Evaluation

Most approaches for quantitative software architecture evaluation only focus on one quality attributes (e.g. performance [10] or reliability [9]). Some qualita-

tive approaches such as ATAM specifically trade off quality attributes based on architecture insights.

In ATAM, the main steps with respect to quality requirement prioritization are the following. In step 2, the business drivers, among them main quality attributes, are discussed and defined. In step 5, a utility tree is defined for quality attributes which capture the importance of quality requirements and the value of achieving a certain level of quality. Thus, the utility tree is a form of quality requirements prioritization. Then, in step 6, possible architectural approaches are evaluated with respect to this utility tree, e.g. by using performance prediction techniques based on queuing networks. Trade-off points where quality attributes conflict with each other are highlighted. However, ATAM does not explicitly support the architect and stakeholders to question and revise the previously defined utility tree based on the evaluation results, but rather focuses on the effects of architecture decisions to find a combination of decisions that together optimize the given utility tree. Our approach complements ATAM by enabling *systematic* feedback for revising the utility tree after architecture evaluation.

Recently, approaches to help the software architect to improve a given software architecture model have been proposed (e.g. PerOpteryx, ArchE, Performance Booster, Archeopteryx [11]). Such approaches automatically vary a given architectural model based on predefined degrees of freedom, such as component allocation to servers, component selection, change of hardware and software parameters, or other, custom defined design decisions expressed as simple model transformations. The reached variants of the architecture are called architecture candidates and are evaluated using multiple quantitative quality prediction techniques. Thus, the approaches explore a part of the design space. Still, so far these approaches only provide feedback to the software architect, and their connection to decisions on the requirements side remains unexplored. In this work, we address the question how to feed the gained information back to the requirements engineering phase.

2.2 Quality Requirements Prioritization in Research

While numerous approaches to handle quality requirements have been suggested [6], few approaches address the prioritization of quality requirements. A survey from 2008 on quality requirements prioritization [8] found that many approaches rely on converting quality requirements into functional requirements first for cost estimation. For example, a security requirement is operationalized to a requirement for a login functionality first.

However, operationalization does not reflect the pervasive nature of such quality requirements as performance or reliability. Furthermore, quality requirements often have the before-mentioned continuous scale, trade-offs among each other, and effect on the utility of each other and the utility of functional requirements [1]. Thus, prioritization techniques for functional requirements are not properly applicable to quality requirements [1, 3].

As an exception, the QUPER approach [4] specifically supports to prioritize quality requirements and supports analysts to define appropriate quality levels.

However, reasoning in QUPER is qualitative and relies on estimating quality costs. Our proposed approach is complementary and could be used to determine QUPER costs barriers and also trade-offs among quality attributes based on quality prediction.

3 Prioritization by Architecture Feedback

Our planned approach provides feedback for requirements prioritization (Fig. 1). Because an initial understanding of quality requirements is required for architecture design, the process starts with the requirements engineering activities and with the design of an initial architecture as before. Compared to previous approaches, more information is collected (design space exploration and analysis of trade-off and dependencies) and a feedback loop from architecture evaluation to requirements prioritization is introduced.

Note that according to Berntsson Svensson et al. [2, 3], the definition of required quality levels is a subtask of requirements prioritization. Quality requirements elicitation is concerned with identifying relevant quality attributes and quality requirements specification is concerned with defining how to measure (or, more generally, test) the quality requirements[1].

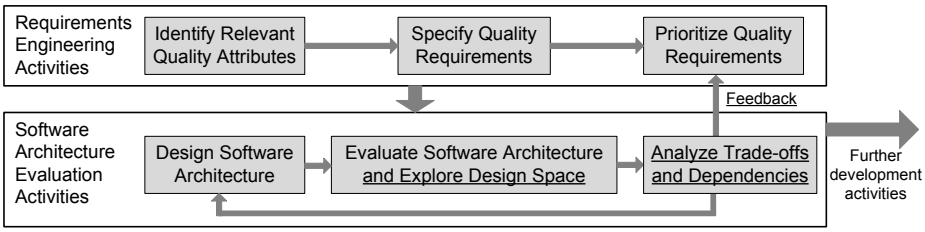

Fig. 1. Prioritizing Quality Requirements using Software Architecture Evaluation (new activities are underlined)

This process can for example be instantiated for a business reporting system (BRS). Only quality requirements are discussed in the following, functional requirements and project requirements are neglected here.

Step 1: Identify relevant quality attributes (stakeholders and requirements engineers): Performance, reliability, and operating costs are relevant for the BRS.

Step 2: Specify quality requirements (stakeholders and requirements engineers): For performance, a response time requirement is defined for the "report-

[1] That is, a quality requirement specification thus only specifies the quality to measure with all its details and environmental conditions (e.g. "the response time of service X must be low under workload Y"), but does not yet define a level of quality (here e.g. "lower than 5 seconds"). If we understood quality level definition as a subactivity of requirements specification instead, Fig. 1 would be changed accordingly and also provide some feedback into the requirements specification phase.

ing" use case. For reliability, the up-time of "reporting" per month is defined. The operating costs are hardware (servers, network, etc.) and maintenance costs.

Step 3: Prioritize quality requirements (stakeholders and requirements engineers): Initially, stakeholders agree that reliability and costs are more relevant than performance. The required quality levels are only roughly defined at this point: The up-time should be as high as economically sensible, while the response time should be low enough that users do not notice waiting times.

Step 4: Design initial architecture (software architect): Based on the initially prioritized quality requirements, the software architect designs an initial architecture and creates an architecture model with quality annotations required for evaluation.

Step 5: Evaluate software architecture and explore design space (software architect and tools): Based on the defined architecture model and existing model-based quality prediction techniques, a design space exploration tool such as PerOpteryx [11] automatically searches the design space for optimal architecture candidates, e.g. by varying component allocation to servers, by changing the hardware to procure, by adding load-balancing or redundancy measures, and by selecting from several available third-party components. Complex architecture models can be handled by such tools, as shown in several case studies [11, 12, 7]. The result is a set of architecture candidates with optimal trade-off between the quality attributes (i.e. *Pareto-optimal* candidates), as shown in Fig. 2. Each point represents a Pareto-optimal architecture candidate and is plotted for the predicted response time and costs of this candidate. Architects can inspect further properties of each found candidate, such as the allocation, with the tool.

Step 6: Analyze trade-offs (software architect): Based on the design space exploration results (Fig. 2), the software architect notes that all three quality attributes are in conflict. Optimal response time and costs form a typical trade-off curve ($\diamond$), but these architecture candidates have a lower availability of 98% per year. To achieve an availability of 99% per year ($\times$), sacrifices for response time and/or costs need to be made. As a result of this step, the discovered quality dependencies and insights are fed back into the requirements prioritization. If more quality attributes are analyzed, advanced tool support from multi-criteria decision support research is required to efficiently explore the found trade-offs.

Step 7: Re-prioritize quality requirements (stakeholders and requirements engineers): Based on the results by the software architect, stakeholders discuss and negotiate on the required quality levels. Finally, they agree that 98% availability is actually sufficient and allows them to achieve a response time of 3 seconds while having low operating costs of less than 500 T EUR

Step 8: Re-design software architecture (software architect): The software architect updates the architecture accordingly by selecting the found optimal architecture candidates just below 500 T EUR. Alternatively, if the stakeholders would not have come to an agreement yet, the software architects could try to make high-level, manual changes to the architecture (e.g. changing the architecture style), and rerun the design space exploration (indicated by the backward arrow to design in Fig. 1).

Step 9: Further development: The architecture design is used to implement the system. The architecture model should be continuously updated, especially with insights for quality properties. For example, the model should be updated by continuous performance measurements of prototypes and first versions of the system. If the quality properties change, the steps above may be revisited.

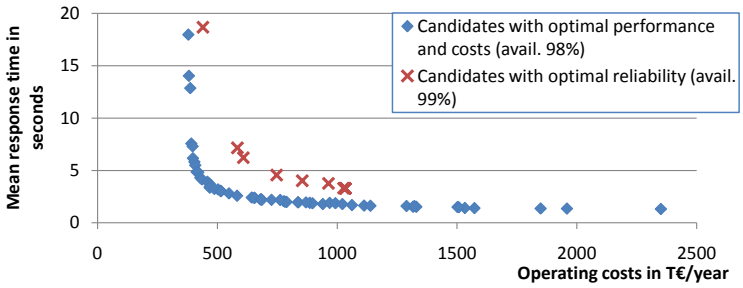

Fig. 2. PerOpteryx Results of BRS Design Space Exploration

As a result, our process supports the iterative and deferred definition of quality requirements, and thus provides a structured approach for stakeholders and software architects to revisit requirements engineering activities after software architecture design.

The design space exploration itself is already realized in the PerOpteryx tool [11] (cf. Sec. 2.1), but no support for interpreting the results (Fig. 2) is available so far. Thus, to support our new process, we will investigate the new step of trade-off and dependency analysis based on design space exploration results as next steps in this research. Here, the main research question is how to extract and represent quality dependencies relevant to stakeholders and requirements engineers, such as conflicts and necessary trade-offs, to support prioritization.

Prioritization by architecture feedback could be applied in more scenarios than the described development process. The prerequisites are (1) that an architecture model of a system is available, and (2) that several quantifiable quality attributes are relevant and can be predicted based on the available architecture model. The architecture model can be (a) an initial architecture model based on initial quality requirements as described above, (b) an initial architecture model based on functional requirements only, (c) a reference architecture for the target domain which is to be adjusted, or (d) the architecture of an existing system which is to be extended or maintained.

4 Conclusion

We present an approach to support quality requirements prioritization by providing feedback from quantitative architecture evaluation and design space exploration. Applying our approach, stakeholders, requirements engineers, and software architects gain a better understanding of the dependencies of quality at-

tributes and the effects of achieving certain quality values. Thus, it helps them to prioritize quality requirements and decide for an optimal trade-off. However, the approach is currently limited to quantitatively evaluated quality properties.

As next steps, we will investigate how the dependencies of quality properties can best be extracted from design space exploration results and how the insight can best be presented to the stakeholders, especially if more than three quality requirements are present.

References

1. P. Berander and A. Andrews. Requirements prioritization. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 69–94. Springer Berlin Heidelberg, 2005.
2. R. Berntsson Svensson, T. Gorschek, B. Regnell, R. Torkar, A. Shahrokni, and R. Feldt. Quality requirements in industrial practice – an extended interview study at eleven companies. *Software Engineering, IEEE Trans. on*, preprint:1, 2011.
3. R. Berntsson Svensson, T. Gorschek, B. Regnell, R. Torkar, A. Shahrokni, R. Feldt, and A. Aurum. Prioritization of quality requirements state of practice in eleven companies. In *RE'11*, pages 69–78. IEEE, 2011.
4. R. Berntsson Svensson, Y. Sprockel, B. Regnell, and S. Brinkkemper. Setting quality targets for coming releases with QUPER: an industrial case study. *Requirements Engineering*, pages 1–16.
5. B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61 –72, may 1988.
6. L. Chung and J. C. S. do Prado Leite. On non-functional requirements in software engineering. In *Conceptual Modeling: Foundations and Applications*, volume 5600 of *LNCS*, pages 363–379. Springer, 2009.
7. T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek. An industrial case study of performance and cost design space exploration. In *ICPE'2012*, Boston, USA, 2012. to appear.
8. A. Herrmann and M. Daneva. Requirements prioritization based on benefit and cost prediction: An agenda for future research. In *RE'08*, pages 125–134. IEEE, 2008.
9. A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and System Modeling*, 7(1):49–65, 2008.
10. H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010.
11. A. Martens, H. Koziolek, S. Becker, and R. H. Reussner. Automatically improve software models for performance, reliability and cost using genetic algorithms. In *WOSP/SIPEW '10*, pages 105–116, New York, NY, USA, 2010. ACM.
12. I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835 – 846, 2011.
13. B. Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–117, 2001.
14. E. Woods and N. Rozanski. How software architecture can frame, constrain and inspire system requirements. In P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrk, editors, *Relating Software Requirements and Architectures*, pages 333–352. Springer Berlin Heidelberg, 2011. 10.1007/978-3-642-21001-3_19.