Architecture-Driven Quality Requirements Prioritization

Anne Koziolek
University of Zurich, Switzerland
koziolek@ifi.uzh.ch

***Abstract*—Quality requirements are main drivers for architectural decisions of software systems. However, in practice they are often dismissed during development, because of initially unknown dependencies and consequences that complicate implementation. To decide for meaningful, feasible quality requirements and trade them off with functional requirements, tighter integration of software architecture evaluation and requirements prioritization is necessary.**

In this position paper, we propose a tool-supported method for architecture-driven feedback into requirements prioritization. Our method uses automated design space exploration based on quantitative quality evaluation of software architecture models. It helps requirements analysts and software architects to study the quality trade-offs of a software architecture, and use this information for requirements prioritization.

I. Introduction

Quality requirements (QRs) drive software architecture design. To study whether a software architecture design will be able to meet given QRs, software architecture evaluation research provides methods [1], including quantitative methods to predict attributes such as performance [2] and reliability [3].

However, while QRs are defined in many companies upfront, they are often dismissed later [4], [5]. In particular, interdependencies and trade-offs among QRs usually remain unclear, so that they cannot be appropriately defined and prioritized. Major difficulties complicate QR prioritization: First, many quality attributes are pervasive, so that the costs and trade-offs to achieve them are hard to estimate in advance [4, pp. 3, 9]. Second, many QRs need to define a required quality level on a continuous scale, such as a response time of 5 seconds. Due to the continuous scale, the prioritization of QR has the additional dimension of choosing the right quality level [5, p. 74].

A tighter integration of requirements engineering and software architecture has been proposed [6], [7], emphasizing feedback from architecture to requirements. These works describe a mindset for software architects; however, they do not provide concrete methods and tool support to combine the two worlds.

In this paper, we describe a tool-supported method for architecture-driven feedback into requirements prioritization. Our method employs an automated design space exploration technique based on quantitative evaluation of quality attributes of software architecture models. Our methods helps requirements analysts and software architects to study the quality trade-offs of a given software architecture, and use this information for requirements prioritization. In particular, the method helps to (1) decide on required quality levels considering quality trade-offs and costs (QR prioritization), and (2) make trade-off decisions between quality and functionality (prioritization between QR and functional requirements).

The contribution of the paper is the description of a process how to quantitatively support quality requirements prioritization using model-based software architecture evaluation, in particular including the required modelling and analysis steps. A preliminary version of the underlying vision has been published before [8], but did not detail on the additionally needed steps compared to existing design space exploration techniques. We have already implemented several steps of this method in the PerOpteryx tool [9] and validated them in multiple settings, including an industrial case study [10]. A full end-to-end implementation and validation is the next step in this line of research.

In the remainder of this paper, Section II describes the architecture-driven QR prioritization process. Section III discusses how existing research results can support individual steps of the process and derives open research challenges for the complete realization of the process. Finally, Section IV discusses related work and Section V concludes.

II. Architecture-driven Quality Requirements Prioritization

The steps of the architecture-driven QR prioritization process are depicted in Fig. 1. The three phases "Modelling", "Automated Exploration", and "Analysis" are detailed in the following three subsections.

To convey our process, we describe its stepwise application to an example based on a real system. The Business Reporting System (BRS) allows users to retrieve statistical reports about business processes from a database. Fig. 2 shows a condensed excerpt of the architecture model of the BRS visualized using annotated UML diagrams. The components are allocated to four different servers. The architecture model also contains behaviour models, an example for the CoreOnlineEngine component is shown in the lower part.

As the input for our process, let us assume that performance, reliability, security, and costs are relevant quality attributes for the example system. In particular, the *relevant QRs* to prioritize are the required response time of the main use case (using the five system services as shown in Fig. 2)

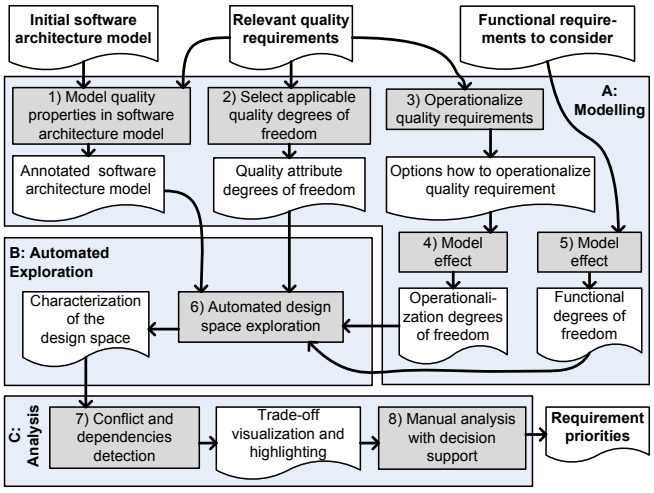

Figure 1. Architecture-driven Quality Requirement Prioritization Process

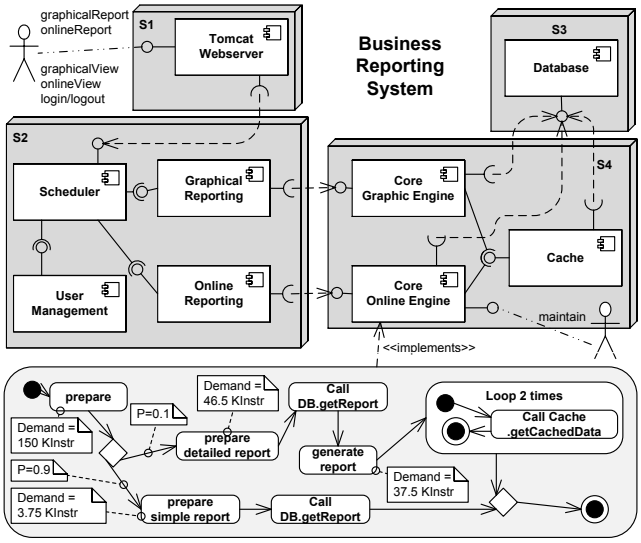

Figure 2. Annotated Architecture Model of the BRS Example. For Example, "Demand" Annotations Describe the Performance Properties of Components

under a given workload, the required probability of failure on demand (POFOD) of this use case, the operating costs, and the protection of the system against unauthorized access. Quality levels are not yet defined. Software architects have designed an *initial architecture*, possibly based on an initial prioritization. Furthermore, requirements analysts want to decide whether to extend the functionality with an automated deviation analysis function, which monitors the business data continuously and sends alarms to the users if deviations in business indicators are recognized (*functional requirement* FR1).

Now, requirements analysts and software architect together want to prioritize QRs, i.e. (1) decide on the required levels of quality for the relevant quality attributes and (2) trade-off with the selected functional requirement.

A. Modelling Phase

First, software architects enhance the given *architecture model with estimated or measured performance and reliability properties* [2], [3] (step 1), resulting in the *annotated software architecture model* (Fig. 2). Hardware nodes are additionally annotated with operating costs estimations, e.g. from vendor specifications (not shown).

The design space to be automatically explored by the tool is modelled as so-called degrees of freedom of the software architecture. Quality degrees of freedom capture possible changes of a software architecture that impact quality attributes. For example, allocating components differently to servers is modelled as an "allocation degree of freedom", expressing the change of the architecture model as a model transformation. Such quality degrees of freedom can be generically identified for a software architecture meta model (e.g. UML) and a quality attribute [11] and thus only need to be *selected* from a library by the software architect [9] (step 2). In the example, software architects select to vary the amount and type of servers to procure as well as the allocation of components to servers, including options where components are replicated to several servers for load balancing and fault tolerance as quality degrees of freedom.

Security is only to be treated by operationalization in this example, without quantitative analysis, because quantitative security evaluation is still ongoing research. Software architects analyze that the weak points of the current architecture is the communication with the user over insecure networks and the resulting accessibility of the servers on this network. Thus, they suggest a dedicated server as a gateway to the insecure network (security option: demilitarized zone server DMZ, step 3). Then, they *model the effects* (step 4) of introducing this option, namely an additional server to which only the uncritical components (Webserver and Scheduler) can be deployed. The resulting degree of freedom is a operationalizing degrees of freedom.

Finally, the effects of the additional functional requirement "FR1: deviation analysis" are *modelled* (step 5), by modelling an additional component called "Deviation Analysis", which accesses the database and is triggered in regular intervals. The resulting degree of freedom to include this components is a functional degree of freedom.

B. Automated Exploration Phase

Combined, the degrees of freedom define a design space for the given architectural model. Then, an *automated search explores this design space* (step 6) by using multi-objective evolutionary optimization [12]. The search subsequently generates candidate architectures in this design space and evaluates them using quantitative quality evaluation techniques.

The objective functions (denoted O_i) are (1) the quantitative evaluations of the mean response time (O_1) and POFOD

Requirements	Effects:
FR1: Deviation analysis	+ $1500 costs *Major, Decision required* + 1 sec RT or combination of both
Security Option: DMZ	+ $100 costs Minor, Negligible

Quality conflicts	Effect Size and Cause
(Performance and reliability) vs. Costs	*Major, Decision required* Main cause: multiplicity of server Minor cause: type of hardware
Performance vs. reliability	Minor, Negligible. Main cause: type of hardware

Figure 3. Example Result Report of the Conflict and Dependency Detection Step for the BRS. Two Conflicts are Highlighted for which Trade-Off Decisions Must be Made (red italics), Two Dependencies are Classified as Minor and Negligible (grey). Possible Causes are Shown.

(O_2) of the system's main use case, (2) the operating costs as derived from the used hardware (O_3), (3) whether to use the security option (O_4), and (4) whether to include the "deviation analysis" (O_5). The latter two objective functions are binary. The search tries to determine the architecture candidates that are optimal trade-offs with respect to these five objectives, i.e. Pareto-optimal[1]. As an approximation of the true Pareto-optimal candidates, the output of the search is a five dimensional Pareto-front [12], which *characterizes* the subset of the design space relevant for trade-off decisions.

C. Analysis Phase

As such a multi-dimensional Pareto-front cannot be directly analyzed by human decision makers (DM) (i.e. requirements analysts and software architects), an automated *conflict and dependency analysis* (step 7) determines the objectives which are actually conflicting so that the DM need to make trade-off decisions. An example report for our example is shown in Fig. 3.

The additional functionality "FR1: deviation analysis" is found to conflict with costs and performance. The dependency analysis finds that this effect is rather constant over the different other degrees of freedom. In the report, it approximates the effect to be $1500 higher operating cost or 1 second longer response time or a combination of both.

For security option 2 "DMZ", only a slight conflict is detected as the exploration finds an architectural candidate with only slightly higher costs by deploying the Webserver and Scheduler components on a dedicated server with a less powerful and thus less expensive configuration, thus tailoring it to the load of these two components. This might be surprising for the DMs, because they might expect higher operating costs because an additional server has to be procured.

However, performance and reliability have a strong conflict with costs. The dependency analysis determines that the use of replication is the cause: If an application server hosting the main components CoreGraphicEngine and CoreOnlineEngine is replicated, this improves both performance and reliability due to load balancing and increased tolerance toward one server failing, respectively. However, the operating costs increase.

[1]Candidate architectures are Pareto-optimal if every further improvement of one objective is only achievable by deteriorating another objective.

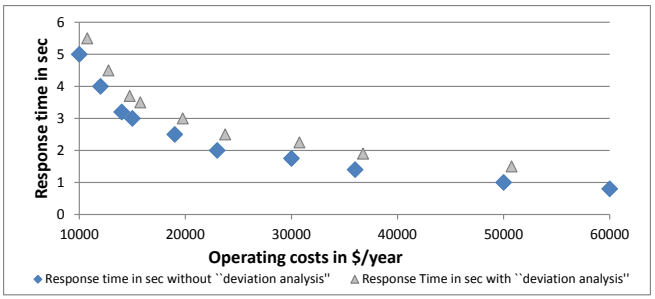

Figure 4. Requested Pareto Front to Inspect (Performance and Reliability) vs. Costs Conflict (with "DMZ" Selected). POFOD Values are Not Shown as They Correlate With Response Time.

To *analyze* this conflict (step 8), DMs can request the relevant view of the Pareto front, assuming that "DMZ" is selected. The resulting view is shown in Fig. 4. The Pareto-optimal architectural candidates are depicted as blue diamonds with their costs and response time values. As performance and reliability correlate, only response time is shown in this view. DMs can trade-off these two qualities with the incurred costs. Furthermore, they can inspect the change if the "FR1: Deviation Analysis" requirement is realized. Graphically, realizing the "FR1: Deviation Analysis" requirement moves the Pareto-front to the right and up, as indicated in Fig. 4 with the grey triangles.

In this example, after negotiations with other stakeholders based on this data, stakeholders decide that the best cost/benefit trade-off is to require a response time of 2.5 seconds maximum and operational costs of $25000 maximum (quality level definition) as well as to include the "DMZ" security option and the "FR1: Deviation Analysis" requirement (requirements selection) because their negative quality effect is acceptable and "worth it". Thus, they prioritize and decide on the requirements.

Note that the modelled artefacts can be reused and updated for subsequent analyses, e.g. additional quality attributes, additional functionality, or any other evolution of the design and later system. Thus, our approach supports an iterative handling of requirements and architecture as envisioned by Nuseibeh [6] and Woods and Rozanski [7].

III. Research Questions and Tool Support

In this section, we will first highlight the existing methods and tools used in the process, and then derive missing research challenges to close the remaining gaps.

First, quantitative quality prediction methods based on architectural models (step 1 in Fig. 1) are available for performance [2] and reliability [3]. Furthermore, several approaches to automatically improve software architecture models (step 6) based on such quality evaluation have been

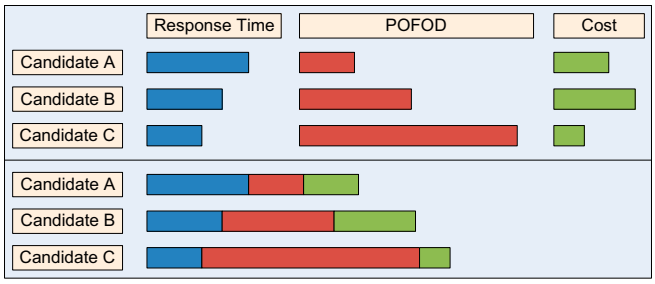

Figure 5. Example Interactive Visualization of Trade-Off Situation

suggested recently. We will realize the process based on our PerOpteryx approach [9], which has already been validated in an industrial setting [10] and supports quality attribute degrees of freedom (step 2).

Finally, approaches from multi criteria decision making (MCDM) [13] will be adopted to help DMs make trade-off decisions satisfying all relevant stakeholders (step 8).

Building on this foundations, we see two research challenges to be tackled by our future research.

On the modelling side, software architecture improvement approaches such as PerOpteryx need to be extended. Currently, quality degrees of freedom are defined on the software architecture metamodel level, capturing general options how to affect quality attributes such as allocation of components to servers. However, to express QR operationalization options (step 4) and functional requirements realization options (step 5), the degree of freedom model will be extended to allow modelling of options on the specific system level.

On the analysis side, the detection of dependencies and conflicts (step 7) needs to be automated to make the insights accessible for DMs. In addition to reports like shown in Fig 3, the output of such a step could be QR dependencies as e.g. used in the Non-Functional Requirements Framework [14] but enhanced with quantitative data. Then, the identified conflicts and trade-offs to be made need to be presented to the DM (step 8). In initial work in this direction, we have explored using the Value Charts [15] MCDM technique as shown in Fig. 5, which visualizes the available candidates and thus helps DMs to interactively explore their own preferences (i.e. change weights of objectives represented by column width) and finally make trade-off decisions.

IV. Related Work

Boehm identified the need for iterative handling of requirements and software architecture decades ago [16]. Nuseibeh's Twin Peaks model [6] suggests to concurrently develop requirements specification and architecture by using insight from one activity in the other. Woods and Rozanski [7] describe how insights 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.

Most approaches for quantitative software architecture evaluation only focus on one quality attributes (e.g. performance [2] or reliability [3]). Some qualitative approaches such as ATAM [17], [1] specifically trade off quality attributes based on architecture insights, however, they do not provide feedback based on quantitative data.

Recently, approaches helping the software architect to improve a given software architecture model have been proposed (e.g. Archeopteryx [18], ArchE [19], PerOpteryx [9], Performance Booster [20]). Such approaches automatically vary a given architectural model based on predefined degrees of freedom, such as component allocation to servers, component selection, or change of hardware parameters and software parameters. Still, these approaches only provide feedback to the software architect and their connection to decisions on the requirements side remains unexplored.

In the field of requirements engineering, numerous approaches to handle quality requirements have been suggested [21], [22]. However, few approaches address the prioritization of quality requirements and none incorporates software architecture evaluation results. A survey from 2008 on quality requirements prioritization [23] found that many approaches only convert quality requirements into functional requirements for cost estimation. For example, a security requirement might be operationalized by requiring a login functionality. However, operationalization does not reflect the pervasive nature and continuous scale of such quality requirements as performance or reliability. Thus, prioritization techniques for functional requirements are not properly applicable to quality requirements in general [24], [5].

As an exception, the QUPER approach [25] specifically supports analysts to define appropriate quality levels. However, reasoning in QUPER is qualitative and relies on manually estimating quality requirements costs without considering complex dependencies. Our proposed approach could complement QUPER by providing these dependencies.

V. Conclusions

In this paper, we have described an architecture-driven QR prioritization process. It helps software architects and requirements analysts (1) to decide on required quality levels (e.g. 5 seconds response time) considering quality trade-offs and costs, and (2) to trade-off quality and functionality.

To realize the proposed process, two main research challenges need to be tackled. First, we will extend our existing software architecture improvement method PerOpteryx [9] to take into account relevant operationalized QRs and functional requirements. The degree of freedom model will be extended to also capture the effects of these requirements. Second, the detection of dependencies and conflicts in the results found by the design space exploration needs to be automated to make the insights accessible for DMs.

REFERENCES

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, Second Edition*. Addison-Wesley, Reading, MA, USA, 2003.

[2] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, pp. 634–658, 2010, special Issue on Software and Performance.

[3] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software and System Modeling*, vol. 7, no. 1, pp. 49–65, 2008.

[4] 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*, vol. preprint, p. 1, 2011.

[5] 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*. IEEE, 2011, pp. 69–78.

[6] B. Nuseibeh, "Weaving together requirements and architectures," *IEEE Computer*, vol. 34, no. 3, pp. 115–117, 2001.

[7] E. Woods and N. Rozanski, "How software architecture can frame, constrain and inspire system requirements," in *Relating Software Requirements and Architectures*, P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrík, Eds. Springer Berlin Heidelberg, 2011, pp. 333–352.

[8] A. Koziolek, "Research preview: Prioritizing quality requirements based on software architecture evaluation feedback," in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, B. Regnell and D. Damian, Eds., vol. 7195. Springer Verlag Berlin Heidelberg, 2012, pp. 52–58.

[9] 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*. New York, NY, USA: ACM, 2010, pp. 105–116.

[10] T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek, "An industrial case study of performance and cost design space exploration," in *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering (ICPE 2012)*, L. Kurian John and D. Krishnamurthy, Eds., Boston, USA, 2012, iCPE Best Industry-Related Paper Award.

[11] A. Koziolek and R. Reussner, "Towards a generic quality optimisation framework for component-based system models," in *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, ser. CBSE '11, I. Crnkovic, J. A. Stafford, A. Bertolino, and K. M. L. Cooper, Eds. New York, NY, USA: ACM, New York, NY, USA, Jun. 2011, pp. 103–108.

[12] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Chichester, UK: John Wiley & Sons, 2001.

[13] V. Belton and T. J. Stewart, *Multiple criteria decision analysis - an integrated approach*. Springer, 2002.

[14] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Springer, 2000.

[15] J. Bautista and G. Carenini, "An integrated task-based framework for the design and evaluation of visualizations to support preferential choice," in *AVI'06*. New York, NY, USA: ACM, 2006, pp. 217–224.

[16] B. W. Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[17] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 1998)*, aug 1998, pp. 68 –78.

[18] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya, "Archeopterix: An extendable tool for architecture optimization of AADL models," in *Proceedings of the 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES 2009)*. IEEE Computer Society, 2009, pp. 61–71.

[19] F. Bachmann, L. Bass, M. Klein, and C. Shelton, "Designing software architectures to achieve quality attribute requirements," *IEE Proceedings - Software*, vol. 152, no. 4, pp. 153–165, Aug. 2005.

[20] J. Xu, "Rule-based automatic software performance diagnosis and improvement," *Performance Evaluation*, vol. 67, no. 8, pp. 585–611, 2010, special Issue on Software and Performance.

[21] L. Chung and J. C. S. do Prado Leite, "On non-functional requirements in software engineering," in *Conceptual Modeling: Foundations and Applications*, ser. Lecture Notes in Computer Science, A. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. K. Yu, Eds., vol. 5600. Springer-Verlag, Berlin, Germany, 2009, pp. 363–379.

[22] M. Glinz, "On Non-Functional Requirements," in *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE 2007)*, 2007.

[23] A. Herrmann and M. Daneva, "Requirements prioritization based on benefit and cost prediction: An agenda for future research," in *Proceedings of the 2008 IEEE 16th International Requirements Engineering Conference (RE 2008)*. IEEE Computer Society, 2008, pp. 125–134.

[24] P. Berander and A. Andrews, "Requirements prioritization," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Springer Berlin Heidelberg, 2005, pp. 69–94.

[25] R. Berntsson Svensson, Y. Sprockel, B. Regnell, and S. Brinkkemper, "Setting quality targets for coming releases with QUPER: an industrial case study," *Requirements Engineering*, pp. 1–16, 2012, to appear.