How the Understanding of the Effects of Design Decisions Informs Requirements Engineering

Zoya Durdik
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
zoya.durdik@kit.edu

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

Ralf H. Reussner
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
reussner@kit.edu

***Abstract*—Requirements are usually one of the main drivers for software architecture. Although current research acknowledges the opposite effects of design decisions on requirements engineering, it does not go beyond the general idea of their existence. The contribution of this paper lies in the explicit discussion of the effects of design decisions on requirements engineering. We define two types of design decisions and discuss their effect on requirements, and in particular on elicitation and prioritisation. Furthermore, we propose and demonstrate on an example two channels from architectural design to requirements that can be used to drive requirement elicitation and prioritization. This is the base for a new approach where also the results of the quantitative analysis of the effects of requirements on architecture are fed back into the requirements process.**

***Index Terms*—Software architecture, requirements engineering, design decisions**

I. Introduction

The common way to design software is to refine gradually requirements into software architecture. Typically requirements are incomplete, and the later the relevant requirements are elicited, the more expensive their consideration might become [1]. That is why the most recent research is concerned with the understanding of the interaction between software architecture and requirements [2], [3], [4], [5], and, for example, Nuseibeh [6] proposes to refine requirements and architecture iteratively together. However, this understanding is currently only partial, merely qualitative in nature, and does not go beyond the general idea that design decisions (we use the definition by Jansen et al. [7]) can affect requirements engineering. To the best of our knowledge, there are neither a common understanding of these effects, nor available approaches to utilise design decisions in requirements engineering.

This paper aims at setting the direction for a new line of research in a common and quantitative understanding of the effects of architecture on requirements and how this understanding can be used in requirements engineering. To initiate the research in this direction, we propose to distinguish between two types of design decisions: *recurring* and *project-specific*. A *recurring design decision* is a decision about reusing existing design solutions, such as design patterns, software components or Web services, and about deployment of components and of Web services. The *project-specific design decisions* are other decisions that do not fall into the first category, e.g. implementation of a new component.

While project-specific design decisions are hard to deal with in a systematic way, we discovered that the recurring design decisions have a high potential to contribute to and to drive requirements engineering. Our argument is based on two ideas: (a) After an analysis, recurring design decisions can be annotated with decision-specific questions. These decision-specific questions are used to stimulate the software engineer to inquire additional information which is used to justify the design decision. The additional information is elicited by the requirement engineer. (b) Taking recurring design decisions can be semi-automated by using a design space exploration approach. This can lead to the quantitative support of taking trade-off decisions (in particular, trade-off between costs and quality properties or of two antagonistic quality properties). Making such trade-offs explicit and giving quantitative data on the impact of a design decision supports the prioritisation of requirements.

Therefore, we identify the following types of effects of design decisions on requirements: (a) *Elicitation* - additional requirements are needed to take the decision, and (b) *Prioritisation* - the priorities of the requirements have to be adjusted according to their impact on costs and quality properties.

The contribution of this paper is an initial description of a new approach that supports requirements elicitation and prioritisation. This approach explicitly and quantitatively investigates the effects of design decisions on requirements engineering (how can the architecture influence requirements). This results in the identification of two channels from architectural design to requirements — recurring design decisions and design decision space exploration.

II. Channels from Architecture to Requirements

A. How Recurring Decisions Help to Elicit Requirements

Software architecture typically implies a number of recurring design decisions, such as selecting design patterns or components, which we propose to use to drive the requirements activities. The idea is to annotate the recurring design decisions with special checklists – the decision-specific questions – and to store them in a catalogue for future use, as initially proposed for design patterns in our previous work [8]. The questions reflect basic properties of the recurring decision, such as goals, benefits and consequences, and shall support

TABLE I
AN EXCERPT OF THE CHECKLIST FOR THE FAT CLIENT PATTERN

Question
Q1 (G). Would you like a client to be able to perform the functionality in circumstances of potential disconnection to the main server or service?
Q2 (I). Would you like to reduce the load on your main server or network through the higher processing and capacity demands to the client devices?
Q3 (I). Is working offline essential for your application?
Q4 (C). Will the application be running on powerful devices and porting to low-performance devices can be excluded in the future?
Q5 (C). Is your infrastructure limitedly heterogeneous and this is unlikely to change in the future?
Q6 (C). Is potential slower start-up of the application acceptable?

the critical evaluation of the feasibility of the recurring design decision. However, they are not intended to help with the initial selection of the best solution as compared to an expert system.

In Tab. I, we present an excerpt of such a catalogue entry for the Fat Client pattern [9]. Each question has a type: *Goal* (G) - the main goal of the decision, *Intent* (I) - intended features and properties of the decision, and *Consequence* (C) - possible (negative) consequences of the decision.

By checking the questions, a person making the decisions, e.g. a software engineer, receives hints about the design decision in question and its aspects that might have been forgotten or might not have been considered otherwise. These questions and answers are aligned to requirements of the system. If the currently available requirements are not sufficient to answer the questions, the requirements engineer can elicit the additional requirements that are needed at the current stage of the project. The requirements engineer can decide to modify or re-prioritize the existing requirements, if the software engineer discovered contradictions and inconsistencies with the decision. Hereby, the link from the recurring design decisions back to the requirements gets established, thus, driving requirements engineering through architecture.

B. How Design Space Exploration Helps to Prioritize Requirements

Design space exploration is an approach that uses design models and quality evaluation functions to automatically search a given, defined design space for optimal designs. In the course of the design space exploration, the design models are varied along defined degrees of freedom (i.e. search variables). Each so generated design candidate can then be quantitatively evaluated using the quality evaluation functions. If several quality evaluation functions are of interest, the goal of the exploration often is to find the Pareto-front of optimal trade-off solutions, from which a human designer can select one.

For software architecture design, several approaches exist to predict quality properties, such as performance [10], reliability [11], and costs. Approaches have been suggested to leverage such quality prediction functions to explore the design space of a given architecture (e.g. [12], [13], survey in [14], example industrial application in [15]). When exploring, the approaches vary recurring design decisions such as component deployment, component selection, or hardware selection. As these considered recurring design decisions usually do not encode foundational decisions of the software architecture (such as the used architectural style or the fundamental structure of the system), they can be thought of as a form of architecture configuration as opposed to far-reaching design decisions made in core architectural design.

We propose to leverage such design space exploration not only to make decisions for the explored degree of freedom variables (such as component deployment, component selection, hardware selection), but to use it also for valuable feedback for requirements engineering.

A possible approach is to use design space exploration to quantitatively characterize the effects of any design option on quality properties. Then, different design options proposed to achieve a requirement can be considered together to quantitatively characterize effects of a requirement itself.

Let us consider an example of a new computationally intensive requirement in a web application, such as the generation of some graphical plots. This new feature could be realized by two different design options: It could be added into the workflow of the application sequentially (causing delay) or its calculations could be done in parallel (on the same or on a different server, thus causing higher utilization and waiting times). Automatically exploring the design space shows that both design options will cause (a) increased response time or (b) higher costs if more servers are procured or a combination of both. This insight can be used to select among the two different design options. Furthermore, the quality properties attainable by all these design options together *quantitatively characterize the quality trade-offs of the requirement itself*. Thus, these insights can be used to prioritize the requirement.

In our example, software architects and requirements engineers (decisions makers hereafter) might consider not to realize the new feature, as they judge that the costs and/or performance drawbacks are too severe compared to the expected benefits of the feature.

In addition, design space exploration is also supposed to help to prioritize quality requirements by defining appropriate quality levels, as envisioned in our previous work [16].

C. Example

We demonstrate the proposed idea on the example of the Business Reporting System (BRS), adapted from [16]. We extend this example with respect to requirements elicitation and include design decisions into the requirements prioritisation part. The system allows users to retrieve two types of reports with different resource demands. It is implemented as a three tier architecture with thin client running on stationary PC connecting to the application tier though a Tomcat Webserver. A high-level view on its architecture is depicted on Fig. 1, where the current client implementation is marked with blue and the changes are marked with green.

Fig. 1. Business Reporting System Example (Green and blue are the XOR client variants)

In the Tab. II we provide an excerpt of requirements and design decisions for the BRS that are relevant for the understanding of the example.

Now consider the evolution scenario triggered by the new requirement R06 (Tab. II): The reports must be also accessible from the mobile devices.

1) Analysis of the Elements Involved into the Recurring Design Decision: The new requirement R06 implies an unstable internet connection, which confronts with the decision D01, which implies that the thin client has to have permanent connection to the server. Thus, analysing the D01 the software engineer elicits a new requirement R07 (possibly with the help of the requirement engineer): The client must also be able to work without stable internet connection.

This new requirement R07 invalidates the design decision D01, as the fat client would be a more appropriate solution. Using the decision annotations for the fat client (provided in Tab. I), the software engineer can elicit some additional requirements, such as: R08. The device must have sufficient computing resources. Moreover, the recurring design decision not only forwards the elicitation of the new requirement R08, but also triggers the re-prioritisation of the existing ones. The fat client would require regular updates (Tab. I), and thus, either the R06 has to be invalidated, or the priority for the R01 requirement has to be changed (it has to be invalidated). This example demonstrates the potential of single recurring design decisions to drive requirements engineering.

2) Design space exploration: The main available degree of freedom to affect costs and performance of the BRS is the component allocation and the hardware selection. We assume that three different server types are available from the organization's hardware provider, with different capacity and operating costs. Furthermore, we assume that information to evaluate performance such as component resource demand and workload are modelled (cf. [10] for details).

The new fat client design option needs less processing on server side, because fat clients only requests a raw report from the system and do not require the generation of plots. To explore the consequences of this design option, the software architect models its effects on the software architecture by introducing an alternative fat client component that accesses a new service "rawReport" by the webserver (marked green in Fig. 1). This service has lower resource demands than the previous "graphicalReport" service. Then, this design option can be included into the design space exploration.

The design space exploration automatically searches the design space spanned by these three aspects. Based on the resulting trade-off curves shown in Fig. 2, decisions makers immediately see that if the thin client design option is used (blue diamond shapes), requirements R03 and R04 (Tab. II) are in conflict: Only one of them can be fulfilled in the considered subset of the design space. The cheapest architecture candidate that fulfils the response time requirement R03 has operating costs are 15000 per year and thus exceed the planned budget (marked (a) in Fig. 2), whereas the fastest architecture candidate that fulfills the cost requirement R4 has an average response time of 3.5 seconds (marked (b)). If the thin client design option is chosen, there are two options how to resolve the performance-costs conflict: One possible resolution of the conflict is to relax the response time requirement to 3.5 seconds and another solution is to relax the cost requirements to a operating costs budget of 15000.

However, decisions makers also observe that both R03 and R04 can be met if the fat client design option is chosen (green triangle shapes). Thus, using design space exploration, an additional argument for the fat client solution has been found and can be taken into account when re-prioritizing the system requirements. Inspecting the Pareto-optimal results they found, software architects notice that with the fat client design option, an allocation of components to three servers is sufficient to satisfy the response time requirement R03. With the thin client design option, four servers would be required to meet R03, which leads to higher operating costs.

Notice that only using design space exploration, the effect that a server can be saved becomes visible immediately if the fat client solution is used. Without design space exploration, the capacity planning might have been done earlier, and the effect of the fat client solution on the performance-costs trade-off might remain unnoticed.

Finally, decisions makers see that the operating costs could be decreased to 7500 EUR/year if the response time requirement is relaxed to 3.5 seconds (due to the possibility to buy less powerful servers, marked (c) in Fig. 2).

To summarize, decisions makers can prioritize among the conflicting requirement for an easily updateable client (R01, realized by thin client) and for mobile clients (R06) which works without stable internet connection (R08, realized by fat client). Furthermore, they can make additional quality level decisions.

TABLE II
EXCERPT FROM REQUIREMENTS, DESIGN DECISIONS AND NEW REQUIREMENTS (CHANGE RREQUESTS) FOR THE BUSINESS REPORTING SYSTEM

A. Requirements
R01. The client side must require minimum updates (1 per month)
R02. Data integrity shall be warranted in 95% of requests
R03. The response time of the "report" service shall be lower than 3 seconds on average
R04. The operating costs of the system shall be lower than 10000 EUR per year
R05. ...
B. Decisions
D01. The client for the desktop computer is implemented as Thin Client
D02. The Application tier is deployed on four servers.
D03. ...
C. New Requirements
R06. The reports must be also accessible from the mobile devices
R07. ...

III. RELATED WORK

The influence of existing architecture and reusable elements on requirements has been confirmed in several studies, such

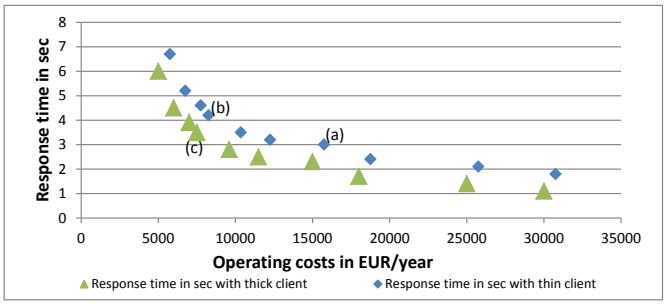

Fig. 2. Results of Design Space Exploration: Trade-off Curves

as in Ferrari et al. [17], or Boer et al. [4]. However, besides observing this influence, they do not provide an approach to utilize architecture or its elements for requirements engineering. Engelsman et al. [18] focus on reverse-engineering of requirements using the architecture and architecture-based requirements specification reuse. The idea to use architecture as a basis for further requirement discovery and determination of the alternative design solutions was first presented by Nuseibeh in [6] and Woods et al. [3]. We build upon these ideas to propose an approach that uses the effects of design decisions on requirements to support requirements elicitation and prioritisation.

In previous work, we have briefly discussed the possibility to use the design patterns for the elicitation of non-functional requirements [8]. In this paper we introduce the idea to use recurring design decisions for the requirement elicitation and prioritisation, where the design patterns are a subclass of such decisions.

Petrov et al. [19] also propose to integrate decision analysis into requirements engineering. The authors deal with specific information sources that can contribute to requirements specification – contextual environment concerns and architectural patterns and heuristics (architectural patterns are not the design patterns [20] we refer to in this paper, but are a kind of "macro-architectural" best practices). These additional information sources can be used complimentary to our approach, as we do not conciser "macro-architectural requirements" [19] explicitly.

We have also discussed the relationship of design space exploration and quality requirements prioritization [21] and proposed a method to systematically support quality requirements prioritization [16]. This paper extends on the previous works focusing on the interplay of design space exploration with other design decisions and general requirements decisions, i.e. considers more than quality requirements.

In the area of software architecture optimization, a large number of approaches have been suggested to improve a given design with respect to several quality properties at once. One type of approaches use multi-criteria optimization (survey in [14]). Another type of approaches uses rule systems to improve a given starting point architecture. For example, the ArchE design assistant [22] applies rules (named tactics) to improve modifiability and performance. However, none of the approaches discusses the feedback that multi-criteria optimization (i.e. design space exploration) can give to the requirements decisions of other than quantifiable quality requirements: They do not support analysing trade-offs between quantifiable quality (such as performance or costs) on the one hand and other quality requirements or functional requirements (such mobile device support in our example) on the other hand.

In the area of quality requirements prioritization and software architecture, methods like ATAM [23] help designers to uncover quality requirements conflicts and find appropriate trade-offs. However, the method is qualitative. While architecture evaluation approaches (e.g. for performance) are mentioned and can be included for single architecture candidates as needed, the relation to automated design space exploration is not discussed.

IV. Further Directions

The effects of the design decisions on requirements engineering need to be investigated further on. We need to empirically evaluate the extent of the feedback of annotated recurring design decisions back to requirements. We need to investigate the benefits of the annotated decisions catalogue on several architectural tasks, such as adding functionalities and refactoring. This should result in a fine-grain classification of design decision types (e.g. deployment or pattern selection) and their effects on requirements.

On the design space exploration side, design options need to be modelled on the architecture level (as shown in Figure 1 in green or blue). Furthermore, design options need to be traced to requirements to reflect the conflicts detected at the design decision level back to the requirements level.

On the other hand, interpretation support for situations with many requirements, design options, and quality evaluation functions need to be devised. Here, conflicts where human decision makers have to make trade-off decisions could be highlighted automatically whereas decisions with small impact are deferred or even decided automatically.

V. Conclusion

In this paper we described a new approach that utilizes the effects of design decisions on requirements engineering to support requirements elicitation and prioritisation. We defined two types of design decisions, and discussed their effects on requirements (partially also including quantitative effects). We identified two channels from architectural design to requirements and a way to use them for requirement engineering activities, which was demonstrated on an example.

There is an urgent need of further research on this topic, and in particular: (a) Fine-grained classification of effects of decisions, and (b) classification of decisions based on their effect on requirements. Such deepened understanding of the effects of design decisions on requirements could be used in an architecture-driven requirements engineering approach to mitigate the problems caused by incomplete requirements specifications.

ACKNOWLEDGEMENT

This work is supported by DFG under the grant number GZ RE 1674/8-1. The authors are thankful for this support.

REFERENCES

[1] P. Gruenbacher, A. Egyed, and N. Medvidovic, "Reconciling software requirements and architectures with intermediate models," in *Software and Systems Modeling*. Springer, 2004, pp. 235–253.

[2] Y. Wu, D. Zowghi, X. Peng, and W. Zhao, "Towards understanding requirement evolution in a software product line an industrial case study," in *Twin Peaks of Requirements and Architecture (Twin Peaks), 2012 IEEE First International Workshop on the*, sept. 2012, pp. 7 –14.

[3] E. Woods and N. Rozanski, "How software architecture can frame, constrain and inspire system requirements," in *Relating Software Requirements and Architectures*. Springer Berlin Heidelberg, 2011.

[4] R. C. de Boer and H. van Vliet, "On the similarity between requirements and architecture," *J. Sys. Soft.*, vol. 82, 2009.

[5] J. Miller, R. Ferrari, and N. Madhavji, "Architectural effects on requirements decisions: An exploratory study," *Seventh Working IEEE/IFIP Conf. on Software Architecture, 2008*, pp. 231 – 240, 2008.

[6] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34 , Issue:3, pp. 115 – 119, 2001.

[7] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, ser. WICSA '05, 2005.

[8] Z. Durdik and R. Reussner, "Position Paper: Approach for Architectural Design and Modelling with Documented Design Decisions (ADMD3)," in *Proceedings of the 8th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2012)*, 2012.

[9] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2005.

[10] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, 2010.

[11] 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.

[12] 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)*, 2009.

[13] A. Martens, H. Koziolek, S. Becker, and R. H. Reussner, "Automatically improve software models for performance, reliability and cost using genetic algorithms," in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, 2010.

[14] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Transaction on Software Engineering*, 2012, preprint.

[15] 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.

[16] A. Koziolek, "Architecture-driven quality requirements prioritization," in *First International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks 2012)*. IEEE Computer Society, 2012.

[17] R. Ferrari, O. Sudmann, C. Henke, J. Geisler, W. Schafer, and N. H. Madhavji, "Requirements and systems architecture interaction in a prototypical project: Emerging results," *16th Int. W. Conf. on Req. Eng.: Foundation for SW Quality*, vol. Volume 6182/2010, pp. 23 – 29, 2010.

[18] W. Engelsman, H. Jonkers, H. M. Franken, and M.-E. Iacob, "Architecture-driven requirements engineering," in *Advances in Ent. Eng. II*, ser. Lecture Notes in Business Inf. Processing, 2009, vol. 28.

[19] P. Petrov, U. Buy, and R. Nord, "Enhancing the software architecture analysis and design process with inferred macro-architectural requirements," in *Twin Peaks of Requirements and Architecture (Twin Peaks), 2012 IEEE First International Workshop on the*, sept. 2012, pp. 20 –26.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Amsterdam, 1995, no. 1.

[21] A. Koziolek, "Research preview: Prioritizing quality requirements based on software architecture evaluation feedback," in *Req. Eng.: Foundation for SW Quality*, ser. Lecture Notes in Comp. Science, vol. 7195, 2012.

[22] A. Díaz Pace, H. Kim, L. Bass, P. Bianco, and F. Bachmann, "Integrating quality-attribute reasoning frameworks in the archE design assistant," in *Proceedings of the 4th International Conference on the Quality of Software-Architectures (QoSA 2008)*, S. Becker, F. Plasil, and R. Reussner, Eds., vol. 5281, 2008, pp. 171–188.

[23] P. C. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures*, ser. SEI Series in Software Engineering. Addison-Wesley, 2001.