2016 IEEE 24th International Requirements Engineering Conference

Towards an Approach to Stimulate Architectural Thinking during Requirements Engineering Phase

Preethu Rose Anish
TATA Research Development and Design Centre (TRDDC), TATA Consultancy Services Ltd., India / University of Twente, Netherlands
preethu.rose@tcs.com

***Abstract*—[Context/Motivation:] The major role of a software architect in the software development life cycle is to design an architectural solution that satisfies the functional and non-functional requirements of the system to the fullest extent possible. However, the details they need to make informed architectural decisions are often missing from the software requirements specifications. Architects therefore either make assumptions or go back to the business analysts for clarifications or conduct additional stakeholder interviews. All of these result in potential project delays. [Question/Problem:] There is very little known about how architects gather this missing information that is crucial to architectural decision-making. Further, there is dearth of knowledge on how we can leverage the requirements analysis phase by empowering business analysts to gather architecturally significant information pertinent to a given requirement. [Principal ideas / Results:] Based on qualitative interview study with software architects from large organizations, this research will (a) investigate how architects cope with missing information in real-world large scale projects and (b) leverage the knowledge of experienced software architects and make it available to business analysts so that they are equipped to elicit a more complete set of requirements that feed sufficient information into the architecture design process. This research further explores the use of machine learning techniques to gauge the potential to introduce automation to the process of gathering the architecturally significant information missing in the software requirement specification.**

***Index Terms*—architecturally significant functional requirements, functional requirements, requirements knowledge, exploratory case study, empirical research method, qualitative interviews, quantitative study.**

I. Introduction and Motivation

Requirements engineering activities often involve capturing both functional and non-functional requirements, describing respectively what the system needs to do, and how it is to be achieved. The software requirements specifications (SRS) created as a result of these activities often lack the details needed by software architects (SAs) to make informed architectural decisions. For example, a SRS might have a requirement "*A notification of all the transactions should be send to the customer*". But details pertinent to what kind of notification is needed (email or a real-time notification) is found missing in the SRS. This is architecturally critical information because depending upon the type of notification, the architecture would change. For email notification, an event driven architecture would suffice as opposed to real-time notification for which a publish-subscribe architecture would be the right choice. As another example, consider a SRS having a high level 'Auditing' requirement, but details pertinent to whether the audit is for internal or regulatory compliance, what data must be produced for compliance, or if notification messages must be sent to stakeholders when certain audit events occur are often found missing in the SRS. Such details impact architectural decisions and should be exposed early on in the requirements elicitation and analysis process.

If wrong architectural decisions are made by virtue of such missing information, the intended but unstated functional requirements will not be satisfied. To compensate, SAs often make unvalidated assumptions based on their own knowledge of the problem or conduct additional formal or informal interviews with the stakeholders to seek clarification and to elicit the missing information. However, assumptions, if incorrect, can lead to expensive and time-consuming refactoring efforts at later stages of projects; while additional unplanned interviews can slow down time-to-market. Asking business analysts (BAs) to provide a richer and more informative specification may seem like a good idea, but it is unlikely to be effective, given that BAs typically lack technical architectural knowledge. They are not equipped to ask the kinds of questions required to extract (from the customer) the information that SAs need in order to make informed architectural decisions. This scenario is typically observed in global software engineering projects, especially the ones of the outsourcing kind where the BA and SA are two separate disciplines and where the two sets of skills - requirements knowledge and architectural knowledge, typically reside with different people.

In the requirements engineering and software architecture literature, it is well understood that both functional and non-functional requirements (NFRs) can have an impact on architectural design. However, the doctoral research proposed in this paper focusses on functional requirements, because (1) the architectural impact of functional requirements is often implicit unlike that of NFRs, which explicitly solicit architectural considerations and (2) much of the published literature [for example: 1 - 6] on architecturally significant requirements has focused on quality concerns (or NFRs) related to security, availability, performance, reliability and other such attributes. Far less emphasis has been placed on addressing the architectural significance of functional requirements [7]. Henceforth, we refer to functional requirements with architectural significance as 'Architecturally Significant Functional Requirements (ASFRs)'. While almost all functional requirements may be argued to have some degree of impact on software architecture; our focus is on those functional requirements that have a **significant** impact. By this, we mean any functional requirement that (i) has a potentially broad impact across the system, (ii) is high-risk, possibly volatile, and (iii) if modified could involve expensive refactoring. By high risk, we mean requirements that are challenging (large risk of failure to implement), constraining or central to the system's purpose (large impact of failure to implement).

The primary goal of this research is to leverage the knowledge of experienced SAs and to make it available to BAs so that they are equipped to elicit a more complete set of requirements that feed sufficient information into the architecture design process.

The remainder of this paper is laid out as follows: Section II provides definitions of the key concepts used in this research abstract, section III provides the research goals and research questions. Section IV provides literature review. Section V details the research method and its execution while Section VI concludes the paper.

II. Definitions

In this section we define the key concepts of this doctoral research abstract.

Architecturally Significant Requirements - Architecturally significant requirements are those requirements that have a measurable impact on the architecture of the software system [33]. Both functional as well as non-functional requirements can fall in the category of architecturally significant requirements. For example, the requirements "*the system shall not be unavailable more than 1 hour per 1000 hours of operation*" and "*An audit trail of all the critical transactions should be maintained*" are examples of architecturally significant requirements.

Architecturally Significant Functional Requirements (ASFRs) – Any functional requirement that is architecturally significant is an architecturally significant functional requirement. "*An audit trail of all the critical transactions should be maintained"* is an example of architecturally significant **functional** requirement.

Architecturally Significant Non-functional Requirements – Any non-functional requirement that has a significant architectural impact is an architecturally significant non-functional requirements. "*The system shall not be unavailable more than 1 hour per 1000 hours of operation"* is an example of architecturally significant non-functional requirement.

Next we define the roles of Business Analyst (BA) and Software Architect (SA) in the context of large global outsourcing projects:

Business Analysts (BAs) – A BA is a role involved in understanding the business or functional aspects of requirements for IT projects or a COTS implementation. The understanding would result in developing detailed functional specifications or gap analysis or review of requirements artefacts or review of developed solutions from a functional completeness perspective.

Software Architect (SA) – A SA is a role involved in converting the business requirements captured by the BA into architecture and design (which then forms the blue print of the project), organizing the development effort, create a vision and consider the project parameters end-to end to attain the vision.

III. Research Goal and Research Questions

The main goal of this research is to define an approach that assists in leveraging the knowledge of experienced SAs and to make it available to BAs so that they are equipped to elicit a more complete set of requirements in terms of architectural details. Based on this objective, the central research question (RQ) of this study is:

How to make the knowledge of experienced software architects available to Business Analysts during requirements elicitation?

To meet the research goal, the main RQ is elaborated in the following sub-questions:

RQ 1: How do SAs currently identify architecturally significant information from a SRS?

- ***RQ 1.1:*** What are Architecturally Significant Functional Requirements (ASFRs) and what is currently known about them?
- ***RQ 1.2:*** What categories of ASFRs have implicit architectural impact for SAs?
- ***RQ 1.3:*** From a SA's perspective, what mechanism is used to unearth the unspecified architectural details in a SRS?
- ***RQ 1.4:*** Having found answer to RQ 1.3, what kind of information is discoverable using the mechanism (identified in RQ 1.3)?

RQ 2: How to use the mechanism to equip BAs to ask more architecturally relevant information? How could we design an approach that integrates the mechanism in an organization's software delivery process?

RQ 3: Is the proposed way of working usable and useful in practice?

- ***RQ 3.1:*** Can practicing SAs use the approach?
- ***RQ 3.2:*** Does the proposed approach take less effort overall than the current way of working?
- ***RQ 3.3:*** Does it lead to architecture designs that are comparable in quality to the current way of working?
- ***RQ 3.4:*** For which kinds of systems and development projects can our approach be used?

IV. Related Work

As a part of related work study, this research involved searching and studying relevant literature in the fields of requirements engineering and software architecture. Particular focus was given on work where intersection between the two fields was found. Although in the requirements engineering and software architecture literature, there is much research on architecturally significant non-functional requirements, we do not include it here as our focus is on ASFRs exclusively. To the best of our knowledge, there is no systematic empirical research that directly addresses our RQs.

A lot of work on the intersection of requirements engineering and software architecture design can be found in the literature [8-16]. In 2001, Nuseibeh proposed the Twin Peaks model to depict the interdependencies that exist between requirements and architectural design [17]. Since requirements specifications and architecture design affect and constrain each other, this model emphasized the need to progressively discover and specify requirements while concurrently exploring alternative architectural decisions. Since then, many researchers have focused their efforts on understanding the impact of requirements on architecture [18-20]. There is also active work that investigates traceability from requirements to architectural designs [21, 22], including links between key stakeholder concerns, architecturally significant requirements, important architectural decisions and sections of code where architectural decisions are implemented [21]. Goknil et al., [22] used the semantics of traces, requirements relations and architecture verification techniques to automatically generate and validate trace links between requirements and architecture leveraging model transformation in ATL and term-rewriting logic in Maude. Chen et al., [23] leveraged the relationship between architecturally significant requirements and architectural design decisions to co-develop NFRs and architecture. In particular, their finding that architecturally significant requirements tend to be described vaguely contributed to the motivation for our proposed work. Niu et al. [24] introduced a practitioner-oriented approach to analyze and evaluate architecturally significant requirements for enterprise systems. However, their focus is on NFRs exclusively. Poort et al. [25], proposed a facilitated method, PALM, which elicited business goals and established the link between those goals and the quality attribute requirements for a system under development. It helped to discover missing quality attributes and empowered the SAs to question the necessity of overly stringent requirements by appealing to stakeholder-expressed business goals. Salehin [26] investigated the extent to which requirements information is missing during the software architecture process and the impact of that missing information on system architecture in terms of effort. They found that SAs actively searched for strategies to reduce missing information in the requirements, so that the software architecture was designed with less effort. This highlights the importance and the need for a mechanism to unearth the unstated architecturally relevant information from ASFRs. Ferrari and Madhavji [27] conducted an empirical study to understand the impact of requirements knowledge and experience (RKE) on software architecture activities. Their findings suggest that SAs with RKE perform better than those without. Khan et al. [28] presented an analysis model that categorized requirements dependencies that are architecturally significant in terms of change impact. They reported on an exploratory study based on the change history analysis of a real-life web-based information system in order to gather the most architecturally significant requirements dependencies from their model. Gross and Dörr [20] reported first results of explorative studies aimed at revealing SAs' information needs that should be fulfilled in software requirements specification. They indicated the relevance of certain requirements artifact types to software architecture, such as system responsibilities, data requirements, system functionalities, interactions, technical constraints, stakeholder goals and their suitable representation in terms of notations. Furthermore, these authors studied the variances among SAs in a group, regarding each one's information needs. In their observations, factors such as expertise and individual motivation might influence the information needs of SAs. The authors call for further research into the information needs of SAs and the factors that precondition them. This doctoral paper attempts to respond to this call.

V. Research Method and Execution

This research project adopts a qualitative interview and survey-based research process for the purpose of problem investigation, treatment design and treatment validation. To materialize this project, Design Science Methodology for Information Systems and Software Engineering [29] is selected. Figure 1 depicts the design and engineering cycles adopted by this research. Real-world implementation in the sense of technology transfer is not part of this research project. Any prototypes that would be tested with real-world SAs would form part of treatment validation. The steps followed in this research are given below:

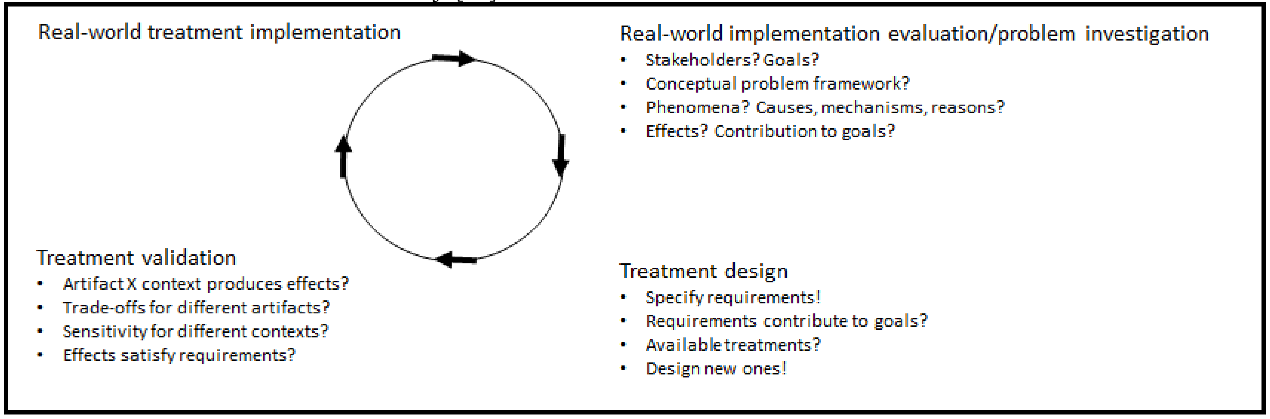
Real-world treatment implementation
Real-world implementation evaluation/problem investigation
• Stakeholders? Goals?
• Conceptual problem framework?
• Phenomena? Causes, mechanisms, reasons?
• Effects? Contribution to goals?
Treatment validation
• Artifact X context produces effects?
• Trade-offs for different artifacts?
• Sensitivity for different contexts?
• Effects satisfy requirements?
Treatment design
• Specify requirements!
• Requirements contribute to goals?
• Available treatments?
• Design new ones!

Fig. 1. Design and engineering cycle used in this research (Wieringa, 2014)

Step 1: The first activity is a structured review of the literature aiming to capture the state-of-the-art in architecturally significant requirements. This produces a number of definitions of concepts that would be used in this research project, as well as a literature survey of what is already known about the problem and what already exists in terms of treatment proposals. This answers RQ 1.1. The findings from this are described in section III.

Step 2: To answer the problem investigation questions – RQ 1 and RQ 2, a qualitative interview based technique is planned by following R. Yin's guideline for exploratory case study design [30]. As part of our research on RQ 1 and RQ 2, we already executed a qualitative case study with 14 practicing SAs from India, the United States and the Netherlands. For the qualitative analysis of the collected data, the guidelines of Charmaz' Constructive Grounded Theory building method [31] was deployed. This approach is used in social sciences to construct general propositions (called a 'theory' in this approach) from textual data. Our study is published in [VII. B]. The key findings of this study are: (1) SRSs often lack crucial architecturally relevant information needed to make informed architectural decisions, (2) Probing Questions (PQs) are an essential mechanism for unearthing underspecified architecturally relevant information, (3) a total of 15 categories of ASFRs, each with its own set of PQs were identified through the study, (4) and there is a logical way to sequence the PQs during the requirements elicitation process.

Step 3: Having identified that the PQs have a logical sequence, we headed towards finding further answer to RQ 2. For this step, we followed a design science approach [29] which was used to discover and document the PQ-flow structures for the identified ASFR categories. We started with a 'basic' design of the PQ-flows and then integrated practitioners' feedback into the design. This resulted in a refined version of the PQ-flows for which further feedback was elicited. The process included the following steps: (a) interview SAs, (b) analyze the PQs obtained through the interviews and create the initial basic PQ-flows in consultation with a SA, (c) conduct an online survey to collect feedback on the early PQ-flows that were created in step (b), refine the PQ-flows in the light of the feedback received, and (d) conduct a follow-up online survey to evaluate the refined PQ-flows from step (c). Based on the analysis of the survey findings, PQ-flows for 5 of the 15 ASFR categories were created. This study is published in [VII. C].

Step 4: To find a more complete answer to RQ 2, we attempted further design by including an automated support for recommending relevant PQ-flows for an SRS and then generating annotated PQ-flows wherein the annotations on the PQs contain possibly relevant requirements from the SRS that already answers some of the PQs. The BAs can leverage this information to determine what is already known, what knowledge is missing, and what effect this would have on the flow of the PQs. For example, if the SRS has the following requirement: "*For regulatory compliance, an audit trail of all the transactions should be maintained*", then there is a clear indication that the audit is for regulatory compliance. In this case, there would be no need to ask the PQ: "*Is the audit for internal compliance or for regulatory compliance?*" For automating this recommendation, we evaluated existing machine learning techniques on our dataset to find if it is feasible to automate the recommendation and annotation of PQ-flows. Initial results on this are reported in [3]. We note that design of machine learning techniques is out of scope for this research. This doctoral research would use some existing machine learning techniques (e.g. Multinomial Naïve Bayes, Vector Space Model) that fit our research purpose.

If the tacit knowledge of the SAs become explicit by means of the PQ-flows and then gets shared with the BAs, we could treat the PQ-flows as pieces of knowledge. In fact, creating a repository of PQ-flows for each ASFR category could generate a body of knowledge similar to the Design Patterns of Gamma et al. [32] and be used by BAs to enhance the quality and completeness of SRSs. To achieve this, the next planned steps in this research are as follows:

- Create PQ-flows for the remaining 10 ASFR categories following a research method similar to the one mentioned in Step 3.
- Conduct replication studies in different domains and different experimental setups. The purpose of this replication study is to help in creation of a knowledge base with ASFRs and their corresponding PQs and PQ-flows. The knowledge base would be managed by a curator who would enter new knowledge in knowledge base, keep the knowledge base well-structured, and support BAs and SAs using and maintaining it.
- This research would also probe into the generalizability of the ASFRs and their corresponding PQs and PQ-flows in the knowledge base. This would help is gauging the role that the curator would have to actually play when this approach is utilized in practice.
- Validate and evaluate the utility and usefulness of the approach. For this, the plan is to do opinion-based and experimental validation. In opinion-based research, we will use focus groups and interviews to collect opinions of practicing BAs and SAs on the usability and utility of the approach. This will be used to further improve the approach. In experimental research, we will test our approach by asking BAs and SAs to perform requirements specification and architecture design tasks using our approach in a context that is as realistic as possible, and compare the results with the current way of working. The most realistic context is where BAs and SAs use our approach in a real project. This would be a Technical Action Research [29]. If this is not feasible, then we will simulate the real world by asking the subjects to use our approach to perform tasks in a realistic but simulated context. This would then be a mechanism experiment.

VI. Conclusion

Software requirements specifications often lack the details needed by the SAs to make informed architectural decisions. SAs therefore either end up making assumptions leading to costly refactoring efforts in later stages of the project development or get back to the concerned stakeholders to gather the missing information resulting in slowing down time-to-market.

In this research abstract, we attempt to devise an approach that leverages the knowledge of experienced SAs and make it available to business analysts so that they are equipped to elicit a more complete set of requirements that feeds sufficient information into the architecture design process. The abstract further explores the amenability of the approach for automation.

VII. Acknowledgment

I am very thankful to my PhD advisor Dr. Roel J Wieringa, PhD co-advisor Dr. Maya Daneva, collaborator Dr. Jane Cleland-Huang and internal advisor Dr. Smita Ghaisas for their continuous guidance, immense support and encouragement for carrying out this research.

VIII. Accepted Publications Related to This Research

A. P.R. Anish, B. Balasubramaniam, J. Cleland-Huang, R. Wieringa, M. Daneva, S. Ghaisas. Identifying Architecturally Significant Functional Requirements, TwinPeaks workshop, ICSE 2015. IEEE Press, 3-8

B. P.R. Anish, M. Daneva, J. Cleland-Huang, R. Wieringa, S. Ghaisas. What You Ask Is What You Get: Understanding Architecturally Significant Functional Requirements, RE 2015, IEEE Press, 86-95

C. P.R. Anish, B. Balasubramaniam, A. Sainani, J. Cleland-Huang, R. Wieringa, M. Daneva, S. Ghaisas, Probing for Requirements Knowledge to Stimulate Architectural Thinking, Accepted at the the 38th International Conference on Software Engineering (ICSE) Austin, May 14 - 22, 2016

References

[1] M. Glinz. A Glossary of Requirements Engineering Terminology. International Requirements Engineering Board, 2011.

[2] N. Niu, L.D. Xu, J.C. Cheng, Z. Niu. Analysis of Architecturally Significant Requirements for Enterprise Systems, IEEE Systems Journal, 8(3), pages 850-857

[3] A. Koziolek. Architecture-driven Quality Requirements Prioritization, TwinPeaks 2012, IEEE Press, pages 15-19

[4] D. Ameller, C. Ayala, J. Cabot, X. Franch. Non-functional Requirements in Software Architecture Practice, report ESSI-TR-12-1, Universitat Politècnica de Catalunya, Barcelona, 2012.

[5] E. Poort, N. Martens, I. van de Weerd, H. van Vliet. How Architects see Non-Functional Requirements: Beware of Modifiability, Requirements Engineering: Foundation for Software Quality (REFSQ'12), pages 37-51.

[6] Cleland-Huang, J., Czauderna, A., & Keenan, Ed., A persona-based approach for exploring architecturally significant requirements in agile projects, Requirements Engineering: Foundation for Software Quality (REFSQ'13), pages 18-33.

[7] N. A. Qureshi, M.B. Usman, N. Ikram. Evidence in software architecture, a systematic literature review. In Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE '13). ACM, New York, NY, USA, pages 97-106

[8] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 3nd ed., Reading, MA: Addison-Wesley, 2011.

[9] D.M. Dikel, D. Kane, and J.R. Wilson, Software Architecture: Organizational Principles and Patterns, Upper Saddle River, NJ: Prentice-Hall, 2001.

[10] C. Hofmeister, R. Nord, and D. Soni, Applied Software Architecture, Boston: Addison-Wesley, 2000.

[11] A. Ran, ARES Conceptual Framework for Software Architecture, in M. Jazayeri, A. Ran, and F. van der Linden, ed., Software Architecture for Product Families Principles and Practice, Boston: Addison-Wesley, 2000, pages 1-29.

[12] M. A. Babar, A. W. Brown, and I. Mistrik. Agile Software Architecture: Aligning Agile Processes and Software Architectures. Morgan Kaufmann, 2013.

[13] J. Bosch, Design and Use of Software Architecture: Adopting and Evolving a Product-Line Approach, Boston: Addison-Wesley, 2000.

[14] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, Boston: Addison-Wesley, 2002.

[15] P. Clements, L. Northrop, Software Product Lines: Practice and Patterns, Boston: Addison-Wesley, 2002.

[16] J. Garland, R. Anthony, Large-Scale Software Architecture: A Practical Guide using UML, Wiley, 2002

[17] B. Nuseibeh, Weaving Together Requirements and Architectures. Computer, 2001. 34(3): pages 115-117

[18] M. Galster, M. Mirakhorli, J. Cleland-Huang, J. E. Burge, X. Franch, R. Roshandel, P. Avgeriou. Views on Software Engineering from the Twin Peaks of Requirements and Architecture. SIGSOFT Softw. Eng. Notes 38, 5, 2013, pages 40-42.

[19] J. Cleland-Huang, R.S. Hanmer, S. Supakkul, and M. Mirakhorli. 2013. The Twin Peaks of Requirements and Architecture. IEEE Software 30, 2, pages 24-29.

[20] Gross, A., Doerr, J. What do software architects expect from requirements specifications? Results of initial explorative studies, Twin Peaks 2012, pages 41-45.

[21] J. Cleland-Huang, Thinking about Quoins: Strategic Traceability of Architecturally Significant Requirements, IEEE Software, vol. 31, no. 4, September/October 2013, pages 16–18.

[22] A. Goknil, I. Kurtev, K. Van Den Berg, (2014), Generation and validation of traces between requirements and architecture based on formal trace semantics. Journal of Systems and Software, 88, pages 112-137

[23] Chen, F. From architecture to requirements: Relating requirements and architecture for better Requirements Engineering, in Requirements Engineering Conference (RE'14), pages 451–455.

[24] N. Niu, L.D. Xu, J.C. Cheng, Z. Niu. Analysis of Architecturally Significant Requirements for Enterprise Systems, IEEE Systems Journal, 8(3), pages 850-857

[25] E. Poort, N. Martens, I. van de Weerd, H. van Vliet. How Architects see Non-Functional Requirements: Beware of Modifiability, Requirements Engineering: Foundation for Software Quality (REFSQ'12), pages 37-51

[26] M.R. Salehin, Missing requirements information and its impact on software architectures: A case study, Master's thesis, The School of Graduate and Postdoctoral Studies, The University of Western Ontario, London, Ontario, Canada, 2013

[27] R. Ferrari, N.H. Madhavji, The impact of requirements knowledge and experience on software architecting: An empirical study, In proceedings of the 6th IEEE/IFIP Working Conference on Software Architecture (WICSA'07), pages 16–25

[28 S. S. Khan, P. Greenwood, A. Garcia, and A. Rashid. On the impact of evolving requirements-architecture dependencies: An exploratory study. In proceeding of 20th International Conference on Advanced Information Systems Engineering (CAiSE'08), volume LNCS 5074, pages 243–257

[29] R.J. Wieringa. Design Science Methodology for Information Systems and Software Engineering. Springer, 2014

[30] R.K. Yin, Case study research, Sage, 2014.

[31] C. Charmaz, Constructing Grounded Theory: A Practical Guide through Qualitative Analysis, Sage, Thousand Oaks, 2006.

[32] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Addison-Wesley: Reading, MA, 1995

[33] L. Chen, M. Ali Babar, and B. Nuseibeh, Characterizing Architecturally Significant Requirements, in IEEE Software, vol. 30, no. 2, pp. 38–45, 2013