Western University

Scholarship@Western

Digitized Theses

Digitized Special Collections

2010

AN EMERGING THEORY ON THE INTERACTION BETWEEN REQUIREMENTS ENGINEERING AND SYSTEMS ARCHITECTING BASED ON A SUITE OF EXPLORATORY EMPIRICAL STUDIES

Remo N. Ferrari

Follow this and additional works at: https://ir.lib.uwo.ca/digitizedtheses

Recommended Citation

Ferrari, Remo N., "AN EMERGING THEORY ON THE INTERACTION BETWEEN REQUIREMENTS ENGINEERING AND SYSTEMS ARCHITECTING BASED ON A SUITE OF EXPLORATORY EMPIRICAL STUDIES" (2010). *Digitized Theses*. 3666.
https://ir.lib.uwo.ca/digitizedtheses/3666

This Thesis is brought to you for free and open access by the Digitized Special Collections at Scholarship@Western. It has been accepted for inclusion in Digitized Theses by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

AN EMERGING THEORY ON THE INTERACTION BETWEEN REQUIREMENTS ENGINEERING AND SYSTEMS ARCHITECTING BASED ON A SUITE OF EXPLORATORY EMPIRICAL STUDIES

(Spine title: Interaction between Requirements Eng. and Systems Architecting)

(Thesis format: Integrated-Article)

by

Remo N. Ferrari

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

© Remo N. Ferrari 2010

THE UNIVERSITY OF WESTERN ONTARIO
SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Supervisor	Examiners
Dr. Nazim Madhavji	Dr. Michael Bauer
	Dr. Steven Beauchemin
	Dr. Matt Davison
	Dr. John Mylopolous

The thesis by

Remo Ferrari

entitled:

An Emerging Theory on the Interaction Between Requirements Engineering and Systems Architecting Based on a Suite of Exploratory Empirical Studies

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date____September 20th, 2010________ ___Dr. Daniel Ansari ______________
Chair of the Thesis Examination Board

Abstract

Requirements Engineering and Systems Architecting are often considered the most important phases of the software development lifecycle. Because of their close proximity in the software development lifecycle, there is a high degree of interaction between these two processes. While such interaction has been recognized and researched in terms of new technology (particularly methods and tools), there is a distinct lack of empirical understanding regarding the scientific properties of this interaction. Furthermore, in Requirements Engineering and Systems Architecting, not only technical but human aspects are considered critical for the success of these processes due to these processes lying at the front-end of the development cycle and therefore being more aligned with real-world issues. Thus, the scientific properties of the interactions between Requirements Engineering and Systems Architecting can be broken down into these two key aspects. For instance, the following example research questions relate to such scientific properties: What is the impact of an existing system's architecture on requirements decision-making? What kinds of requirements-oriented problems are encountered during architecting? What is the impact of an existing systems architecture on new requirements being elicited? What is the impact of requirements engineering knowledge on systems architecting? There is little in the literature addressing such questions.

This thesis explores such issues through a suite of six exploratory empirical studies that were conducted over the last five years. Based on the observations from these studies, an emerging theory is proposed that describes the impact of human and process factors in the interaction between Requirements Engineering and Systems Architecting. The impact of this emerging body of knowledge is deemed to be on the following: technology development for Requirements Engineering and Software Architecting (methods, tools, processes, etc.); hiring and training personnel for Requirements Engineering and Systems Architecture processes in industry; Requirements Engineering and Systems Architecture project planning; curriculum improvement in academia; and future empirical research in Requirements Engineering and Systems Architecting.

Keywords: Software Engineering, Requirements Engineering, Software and Systems Architecture, Empirical Studies, Software Theory, Process Improvement, Software Quality, Social Aspects.

Co-Authorship Statement

In this integrated-article thesis, several of the published papers used as chapters in the thesis were co-authored by the candidate and other authors. Here, the contribution for each of the co-authored chapters is described, including specifically who contributed to the work and the nature and extent of this contribution. Firstly, the contribution of the thesis supervisor, Professor Nazim H. Madhavji, to every thesis chapter will be described. Following this, other chapters that had specific contributions from other co-authors will be given.

All thesis chapters:

Professor Madhavji's contribution to the research described in this thesis, as expected, is that of an involved supervisor. From inception of the idea of the interaction between system requirements and architectures to the facilitation, design and conduct of each study described in thesis, his role has been that of an active, involved, supervisor. Likewise, his supervisory role has been active guidance and feedback in: the decision to carve out the scope for a particular paper from the research being conducted, discussing, writing and editing each paper.

Chapter 4

For this work, the co-author was James Miller, a former Master's student of the University of Western Ontario, who was involved in the design of the study, data collection process and preliminary analysis and interpretations. The preliminary report comprised an aspect of his Master's thesis (completed in April 2006), however, this co-author was not involved in the writing of the published manuscripts as they appear in this thesis, including the significant data analysis and interpretations that were conducted, and substantial literature review and the discussion on the implications of this work.

Chapter 5:

For this work, the co-author was James Miller, a former Master's student of the University of Western Ontario, who was involved in the design of the study, data collection process and early data analysis phases, and early report writing. As in Chapter 4, major re-analysis of this study was carried out from purpose to final report writing, as it appears in this thesis.

Chapter 6:

In this paper, the co-authors involved were Oliver Sudmann, Jens Geisler, Christian Henke and Professor Wilhelm Schafer, all from the University of Paderborn in Germany. These co-authors provided documents for data, validated emergent findings, were interviewees in the study's data collection process, and reviewed report writing.

Chapter 7:

The co-author in this work was Mark Wilding from IBM Canada. He was involved in the research idea formulation phase and also reviewed written reports.

Acknowledgements

It is a pleasure to thank those that made this thesis possible.

First off, I would like to sincerely thank my supervisor Nazim H. Madhavji. It has been an honour to work under his guidance and supervision for the past six years including my Master's and Ph.D. degrees, and I am truly indebted for his support, hard work, dedication and encouragement during this time.

I would like to thank my wife Sav and our kids, Risa and Marcello, for their ongoing patience and support, especially during all the late nights and long days. To my parents, grandmother, and other family members, I appreciate the encouragement you have given to me.

I am grateful to the many colleagues, staff and friends who I have had the opportunity to work or associate with, at both UWO and the University of Paderborn in Germany. In particular, I would like to thank Professor Wilhelm Schafer for hosting me in Paderborn and providing a great collaborative research opportunity with the RailCab project. Also a special thanks to Oliver Sudmann for all his hard work in collecting data, coordinating project logistics and translating documents. As well to Jens Geisler and Christian Henke, for their valuable time in assisting with data collection and validation in the RailCab study. To James Miller, who was a pleasure to work with in our research collaboration at UWO. There are simply far too many people to list here, but I am ever grateful for their assistance and friendship.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was funded by the Ontario Graduate Scholarship in Science and Technology for my first three years, and the Ontario Graduate Scholarship my last two years. NSERC of Canada also supported my work.

Last, but certainly not least, I would like to thank the participants of the various studies; I greatly appreciate the effort that they expended in the research, and hope that the projects were a great learning value for each of them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Requirements Engineering (RE) and Systems Architecting (SA) are often considered the most important phases of the software development lifecycle (IEEE SWEBOK, 2004, Booch 2007). RE encompasses the set of tasks and activities that go into determining the needs or conditions for a software system (Kotonya and Sommerville, 1998). Likewise, SA encompasses those tasks and activities that aid in determining "the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution" (IEEE Standard 1471, 2000).

Because of their close proximity in the software development lifecycle, there is a high degree of interaction between these two processes (Nuseibeh, 2001). For example, when eliciting and analyzing requirements, it is important to assess the impact of certain requirements on the backbone of the system (Kotonya and Sommerville, 1998); likewise, when architecting the system (or part thereof), it may be necessary to elicit new, or refine certain, existing requirements (Nuseibeh, 2001).

While such interaction has been recognized and researched in terms of new technology (methods [Bass et al., 2003][Wang et al., 2005], development methodologies [Castro et al., 2002], tools [Bachmann et al., 2003], processes [Schwanke, 2005][Brandozzi and Perry, 2003], etc.), there is a distinct lack of empirical understanding regarding the scientific properties of this interaction. For example, we do not know the impact of an existing SA on newly elicited requirements, the kinds of requirements-oriented problems experienced while architecting a software system, and the impact of RE knowledge and experience on SA. Such an understanding would (a) add substantially to the body of knowledge in Software Engineering (SE) (IEEE SWEBOK, 2004) and (b) could be used in the design, assessment and improvement of methods, tools and processes (Wieringa and Heerkens, 2006).

Furthermore, in SE, both technical and human aspects are considered critical for the success of software development (John et al., 2005; Bass and Berenbach, 2008; Clements et al., 2007). In particular, for RE and SA, human factors are even more important due to these processes lying at the front-end of the development cycle and therefore being more aligned with real-world issues. Unlike, for example, the testing or coding phases where much of the work can be aided by (semi-) automated technology, RE and SA, due to their inherent activities and tasks, cannot be driven predominantly by technology but are instead heavily dependant on the actual human agents conducting these processes (Nuseibeh and Easterbrook, 2000). Thus, the scientific properties of the interactions between RE and SA can be broken down into two key categories: technical and human based. Technical properties deal with such example issues as: concrete models in RE and SA, traceability between SA elements and requirements, and the choice of architecting tactics and their impact on requirements satisfaction (Bass et al., 2003). Whereas, orthogonal to technical properties, human-based properties deal with example issues such as: the RE competency of software architects and its impact on SA quality, how personal interests and motivation effect RE and SA products, and the importance of non-technical skills (communication, leadership, etc.) in RE and SA activities.

Thus, this thesis explores such issues through a suite of six empirical studies in the RE and SA domains that were conducted over the last six years. The studies were conducted in a variety of contexts, such as academic "lab" experiments, and case study on real large-scale project. The studies also employed a mix of empirical methodological designs, from quantitative based data collection and analysis, to more Social Sciences oriented qualitative techniques (Creswell, 2003). This empirical exploration is ultimately meant to enhance the overall body of empirical knowledge pertaining to the interaction of RE and SA, from both a technical and human-based viewpoint.

1.1 Research Contribution

Based on the observations from these six studies, an emerging grounded theory is proposed that describes the impact of human and technical factors in the interaction between RE and SA. In short, the theory states that:

The effectiveness of RE and SA processes is increased if technological support ensures:

(1) tighter coupling between the artefacts and activities across RE and SA[1]*,*

(2) the variety of the project's development context (such as new development vs. enhancements, agile vs. traditional development models, centralized vs. distributed organization, etc.), and

(3) compatibility with the varying degrees of knowledge, skill-sets and personal motivation possessed.

The emerging theory is subsequently evaluated for its "goodness" based on SE theory-construction guidelines from (Boehm, 2006) and (Sjoberg, 2008). This emerging theory is novel in the RE and SA fields, and provides practitioners with scientific principles regarding key issues in RE and SA, and researchers with an explicit framework for discussing and conducting further RE and SA research.

The impact of this emerging body of knowledge (both the empirical studies findings and emerging theory) is deemed to be on the following: technology development for Requirements Engineering and Software Architecting (methods, tools, processes, etc.); hiring and training personnel for Requirements Engineering and Systems Architecture processes in industry; Requirements Engineering and Software Architecture project planning; curriculum improvement in academia; and future empirical research in Requirements Engineering and Software Architecting. Each of the six conducted studies has their own detailed implications in the above areas.

1.2 The Thesis Core: Six Studies

The thesis core is characterized by the mentioned six studies, one per chapter. Table 1-1 provides a concise overview of all the studies in chronological order,

[1] Technology ensuring tighter coupling between RE and SA process activities does not imply that the processes must be conducted in a more tightly integrated manner; it simply means that this capability should be present in the emplyoyed technology to use as appropriate in the RE and SA processes.

containing the following information: the title of the study, the chapter where the study is described in detail, study date, and any refereed publications resultant from the study.

Chapter #	Study Title	Study Date	Publications
2	Impact of RE knowledge and experience on Software Architecting	2004-2005	(Ferrari and Madhavji, 2008a), (Ferrari and Madhavji, 2007)
3	Requirements-oriented problems while Architecting	2005	(Ferrari and Madhavji, 2008b), (Ferrari and Madhavji, 2006)
4	Requirements characteristics in the presence/absence of an existing SA	2005-2006	(Ferrari et al., 2010c), (Miller et al., 2009),
5	Impact of existing SA on requirements decisions	2006-2007	(Miller et al., 2010), (Miller et al., 2008)
6	Impact of existing SA on requirements decisions in a large-scale, prototypical context	2008-2009	(Ferrari et al., 2010a), (Ferrari et al., 2010b)
7	Impact of non-technical factors' on SA	2009	(Ferrari et al., 2009)

Table 1-1. Overview of studies

We now provide an overview of each study.

1.2.1 Impact of RE Knowledge and Experience on Software Architecting

This study investigated the impact of requirements knowledge and experience (RKE) possessed by the human agents conducting a systems architecting project. Specifically, it describes an exploratory, controlled study involving 15 architecting teams, approximately evenly split between those teams with RKE and those without. Each team developed its own system architecture from the same given set of requirements in the banking domain. The subjects were all final year undergraduate or graduate students enrolled in a university-level course on systems architectures. The overall results of this study suggest that architects with RKE develop higher-quality software architectures than those without, and that they have fewer architecture-development problems than did the architects without RKE.

We also identified specific areas of both architecture design as well as the architecture-development process where the differences manifest between the RKE and non-RKE architects. Implications of the findings are discussed, and the focus on the areas of hiring and training, pedagogy, and technology. The empirical study was carried out using the ''mixed methods" approach, involving both quantitative and qualitative aspects of the investigation. A bi-product of this study is an architectural assessment instrument (included in Appendix A) for quantitative analysis of the quality of a systems architecture.

1.2.2 Requirements-oriented problems while Architecting

Requirements permeate many parts of the software development process outside the RE process. It is thus important to determine whether software developers in these other areas of software development face any requirements–oriented problems in carrying out their tasks. Feedback so obtained can be invaluable for improving both requirements and RE technologies. This study was an exploratory case study of requirements-oriented problems experienced by sixteen architecting teams designing the same banking application. The study found that there were several different types of requirements-oriented problems, of varying severity, which the architects faced in using the given requirements; those architects with RE background also faced requirements-oriented problems; and about a third of all problems were requirements-oriented problems.

Furthermore, there was much concurrence of our findings with software-expert opinion from a large insurance company. We also discuss implications of the findings for the RE field, particularly in the areas of: expression of quality requirements for different stakeholders; empirical studies on quality scenarios; tighter integration of RE and software architecting processes; and requirements to architecture mapping. There are opportunities for further research based on two emergent hypotheses that are also described.

1.2.3 Requirements characteristics in the presence/absence of an existing SA

While much research attention has been paid to transitioning from requirements to systems architectures, relatively little attention has been paid to how new requirements are affected by an existing system architecture. Specifically, no scientific studies have been conducted on the ''characteristic'' differences between the newly elicited requirements gathered in the presence or absence of an existing systems architecture. This study was an exploratory controlled study investigating such requirements characteristics. We identify a multitude of characteristics (e.g., end-user focus, technological focus, and importance) that were affected by the presence or absence of an SA, together with the extent of this effect. Furthermore, we identify the specific aspects of the architecture that had an impact on the characteristics. The study results have implications for RE process engineering, post-requirements analysis, requirements engineering tools, traceability management, and future empirical work in RE based on several emergent hypotheses resultant from this study.

1.2.4 Impact of existing SA on requirements decisions

The question of the "manner in which an existing systems architecture affects requirements decision-making" is considered important in the research community; however, to our knowledge, this issue has not been scientifically explored. We do not know, for example, the characteristics of such architectural effects. We conducted an exploratory study on this question. Specific types of architectural effects on requirements decisions are identified, as are different aspects of the architecture together with the extent of their effects. This study reported quantitative measures and qualitative interpretation of the findings. The understanding gained from this study has several implications in the areas of: project planning and risk management, RE and SA technology, architecture evolution, tighter integration of RE and SA processes, and middleware in architectures. Furthermore, we describe several new hypotheses that have emerged from this study, that provide grounds for future empirical work. This study involved six RE teams (of university students), whose task was to elicit new requirements for upgrading a pre-existing banking software infrastructure. The data collected was based on a new meta-model for requirements decisions, which is a bi-product of this study.

1.2.5 Impact of existing SA on requirements decisions in a large-scale, prototypical context

This study continues the investigation of the study described in the previous subsection (Section 1.2.4) on the impact of an existing SA on requirements decisions. While the findings from that initial study were promising, much work still remains to solidify the results. Therefore, we conducted a replication of the study, and its significant extension, on a large-scale prototypical rail project being developed in Germany. Specifically, we identify (i) the effects of SA on RE decisions, (ii) the characteristics of the RE decisions and (iii), the impact of such decisions on development activities and the rail system. The findings of this study have implications on tighter RE and SA integration across subsystems, impact analysis of requirements on SA, and planning and risk management. We also propose three emergent hypotheses from this case study as a driver for future empirical work in RE. This case study involved examining the 10-year history of requirements and architecting decisions in several major components of the rail project. The data collected was from numerous project documents and extensive interviews with the developers and planners.

1.2.6 Impact of non-technical factors on SA

As discussed in the introduction, most of the research and pedagogical literature in RE and SA are on technical issues. Recently, however, there has been increasing interest on the importance of non-technical factors such as leadership, communication, inter-personal skills, work habits etc. in RE and SA. Despite this, to our knowledge, no empirical studies have been conducted to examine the impact of non-technical factors in Systems Architecture from the viewpoint of academia. In this study, we conducted a multiple-case study where we analysed non-technical problems encountered from 15 student architecting teams to determine the types of problems students have, and also their impact on the quality of the architecture. We found that there were 156 non-technically oriented problems distributed among the teams, and spread among numerous categories of problems. We also found that there was a moderate correlation between a team's number of non-technical problems encountered and the final architecture quality.

Furthermore, we analyzed the IEEE/ACM Software Engineering and Computer Science curriculums to determine any correspondence between these curriculums and the student's architecting performance. Our general finding is that non-technical issues are under-represented in the current curriculums. For example, only 7% of the total hours in a recommended curriculum are allocated to non-technical factors. Based on this analysis we make recommendations for the improved education of student software architects.

1.3 The Thesis Structure

This thesis is documented in the "integrated-article" format[2]; under this format, each discrete study is reported in its own chapter (Chapters 2 to 7) and following these chapters is a chapter describing the emerging theory (Chapter 8), effectively abstracting and relating the various studies under a set of theoretical propositions. Lastly, Chapter 9 concludes the thesis and describes future work[3].

References

Bachmann, F., Bass, L., Klein, M., 2003. Moving from quality attribute requirements to architectural decisions. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 122-129.

Bass, L., Berenbach, B., 2008. Leadership and management in software architecture (LMSA'08). a report on an ICSE workshop. ACM SIGSOFT Software Engineering Notes 33(4): 27-29.

Bass, L., Clements, P., Kazman, R., 2003. *Software Architecture in Practice, 2nd edition*, Addison-Wesley.

Boehm, B., Jain., A., 2006. An Initial Theory of Value-Based Software Engineering. In Value-Based Software Engineering 1st Edition; Eds: Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grünbacher G.; Springer, Germany, pp. 15-33.

Booch, G., 2007. The Irrelevance of Architecture. IEEE Software 24 (3), 10-11.

[2] Official university information pertaining to the "Integrated-article" format can be found at http://grad.uwo.ca/current_students/trg_3.htm

[3] In the integrated-article thesis format, a related work or background chapter is not mandatory. In this thesis, this type of chapter is omitted since each chapter of the thesis has its own extensive related work section, and it was deemed to be superfluous to repeat this information.

Brandozzi, M., Perry, D. E., 2003. From Goal-Oriented Requirements to Architectural Prescriptions: The Preskriptor Process. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 107-113.

Castro, J., Kolp, M., Mylopoulos, J., 2002. Towards Requirements-Driven Software Development Methodology: The Tropos Project. Information Systems, June 2002.

Clements, P. C., Rick Kazman, Mark Klein, 2007. Working Session: Software Architecture Competence. WICSA 2007.

Creswell, J. W., 2003. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* Thousand Oaks, CA: Sage Publications.

Ferrari, R., and Madhavji, N. H., 2006. Requirements-Oriented Problems While Architecting: An Empirical Study. 12th Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 81-96, Luxembourg.

Ferrari, R., and Madhavji, N. H., 2007. The Impact of Requirements Engineering Knowledge and Experience on Software Architecting: An Empirical Study. 6th Working IEEE/IFIP Conference on Software Architecture, Mumbai, India.

Ferrari, R., and Madhavji, N. H., 2008a. Software architecting without requirements knowledge and experience: What are the repercussions?. Journal of Systems and Software (2008), Volume 81 Issue 9, pp. 1470-1490, September 2008.

Ferrari, R. and Madhvaji, N. H., 2008b. Architecting-Problems Rooted in Requirements. Special Journal Issue of Information and Software Technology on Best Papers from REFSQ'05 and '06, Volume 50, Issue 1-2, January 2008, pp. 53-66.

Ferrari, R., Sudmann, O., Geisler, J., Henke, C., Schaefer, W., Madhavji, N. H., "Requirements Engineering Decisions in the Context of an Existing Architecture: A Case Study of a Prototypical Project", In Proceedings of the 18th IEEE Requirements Engineering Conference, Sydney, Australia, 2010, pp. 79-88.

Ferrari, R., Sudmann, O., Geisler, J., Henke, C., Schafer, W., Madhavji, N. H., 2010b. Requirements and Systems Architecture Interaction in a Prototypical Project: Emerging Results. In Proceedings of the 16st International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2010), pp. 23-29, Essen, Germany.

Ferrari, R., Miller, J., Madhavji, N. H., 2010c. A Controlled Experiment To Assess The Impact Of System Architectures On New System Requirements. Requirements Engineering Journal, Volume 15, Issue 2 (June 2010), RE'09 Special Issue; Guest Editor: Kevin T Ryan, pp. 215-233, 2010.

Ferrari, R., Wilding, M., Madhavji, N. H., 2009. The Impact of Non-Technical Factors on Software Architecture. 2nd Workshop on Leadership and Management in Software Architecture, pp. 32-36, Vancouver, Canada.

Guide to the Software Engineering Body of Knowledge: 2004 Version. IEEE and IEEE Computer Society project. <http://www.swebok.org/>.

IEEE Std. 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems Description.

John, M., Maurer, F., Tessem, B., 2005. Human and social factors of software engineering: workshop summary. ACM SIGSOFT Software Engineering Notes Volume 30, Issue 4 (July 2005), pp. 1 – 6.

Kotonya, G., Sommerville, I., 1998. *Requirements Engineering - Processes and Techniques*. Wiley.

Miller, J., Ferrari, R., and Madhavji, N., 2008. Architectural Effects on Requirements Decisions: An Exploratory Study. 7th Working IEEE/IFIP Conference on Software Architecture, pp. 231-240, Vancouver, Canada.

Miller, J., Ferrari, R., and Madhavji, N. H., "An Exploratory Study of Architectural Effects on Requirements Decisions", article in press, doi:10.1016/j.jss.2010.07.006, 2010.

Miller, J., Ferrari, R., and Madhavji, N., 2009. How do System Architectures Affect Software Requirements?. Proceedings of the 31st International Conference on Software Engineering (ICSE '09), pp. 287-290, Vancouver, Canada.

Nuseibeh, B., 2001. Weaving Together Requirements and Architectures. IEEE Comp., March 2001, 34(3): 115-117.

Nuseibeh, B.; Easterbrook, S., 2000. *Requirements engineering: a roadmap*. Proceedings of the Conference on the Future of Software Engineering, ACM Press, pp. 35-46.

Schwanke, R., 2005. GEAR: A Good Enough Architectural Requirements Process. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, USA, pp.57-66.

Sjøberg, D., Dybå, D., Anda, B. C., and Hannay. J., 2008. Building theories in software engineering. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, Guide to Advanced Empirical Software Engineering, pages 312–336, Springer.

Wang, Z, Sherdil, K., Madhavji, N., 2005. ACCA: An Architecture-centric Concern Analysis Method. IEEE Working International Conference on Software Architecture (WICSA), Pittsburgh, USA, November 2005, pp. 99-108.

Wieringa, R. J., Heerkens, J., 2006. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. Requirements Engineering Journal, Vol. 11, pp. 295-307.

Chapter 2

Impact of Requirements Engineering Knowledge on Systems Architecting[4]

1 Introduction

The relationship between Requirements Engineering (RE) and Systems Architecture (SA)[5] has been of significant interest in the past five or so years in terms of methods for designing, transitioning, recovery and analysis; notations and representations; design and analysis tools; development paradigms; and project experiences that aim to facilitate a smoother transition from RE to SA. See, for example, focused workshops on RE and SA transitioning (STRAW 01, STRAW 03) and some individual research works, such as: (Damian and Chisan, 2006; Wang et al., 2005; Schwanke, 2005; Poort et al., 2004; Rapanotti et al., 2004).

However, the relationship between RE and SA in terms of the human agents conducting these processes has not been explored scientifically. It is important to examine the RE and SA relationship in this manner because, not least the fact that, RE and SA processes are typically adjacent to each other in a development project causing substantial interaction between these two processes and that a scientific understanding of the RE and SA relationship would add substantially to the body of knowledge in software engineering (IEEE SWEBOK, 2004).

For example, when eliciting and analysing requirements, it is important to assess the impact of certain requirements on the backbone of the system (Kotonya and Sommerville, 1998); likewise, when architecting the system (or part thereof), it may be necessary to elicit new, or refine certain, existing requirements (Nuseibeh, 2001). Thus, being knowledgeable in these cross-functional areas is an asset when conducting the RE and SA processes. Still, beyond such intuition, no previous studies

[4] A version of this chapter has been published in (Ferrari, R., and Madhavji, N. H., 2008).
[5] For the rest of the paper, the acronym SA will refer to: Systems Architecture as a discipline, an

exist to our knowledge that critically examines the RE and SA relationship in terms of the knowledge and experience of the agents. In this paper, we examine this issue, with a particular focus on the *impact of requirements knowledge and experience* (RKE) *on SA tasks*.

RKE encompasses knowing about, and having experience with, the various technical areas in engineering requirements such as: elicitation, modelling, analysis of requirements, negotiation, prioritisation, quality drivers, viewpoints, specification, validation, traceability, process, management, etc. (Kotonya and Sommerville, 1998). SA, on the other hand, encompasses such tasks as: quality drivers determination, tactics and pattern determination, module decomposition, interface specification, behaviour modelling, documentation of different system views, etc. (Bass et al., 2001).

Some critical questions in the RE and SA relationship include: What is the impact of RKE on (i) the determination of architecting-tactics to satisfy quality requirements, or (ii) the determination of architectural patterns to integrate the architectural tactics and quality drivers, or (iii) formation of architectural abstractions? There are many other such questions and, clearly, knowing the dependency of SA on RE has important implications for such purposes as hiring, training, education and technology in the SA field. In this paper, we describe an empirical study on such issues.

In our study, for example, we found that architects with RKE significantly performed better, in terms of architectural quality, than those without. Specific technical areas where the RKE group seemed to excel were in *tactics, quality satisfaction, pattern determination, module decomposition* and *interface specification.* These findings are not only new but are also surprising because both types of architects, with and without RKE, were trained in the same way on SA tasks and, as one would expect, the SA training inevitably had to address the relevant requirements and architecting issues.

The findings of the study have implications for development practice, education and training. For example, in industry, such findings could feed into

artefact, or the architecting process. The context in which the acronym appears dictates its meaning.

development of new, or improvement of, tools and paradigms that could better help architects without RKE. Also, problem areas that were encountered by both, RKE and non-RKE, architects could be identified and focused upon in the development and tuning of RE and SA methods, tools, procedures and processes. Likewise, in academia, analysis of numerous SA courses offered by various post-secondary institutions suggests that the extent of the RE knowledge required as a pre-requisite to taking SA courses is quite variable. Out of ten respected institutions that offered SA, only one SA course had a dedicated RE course as a prerequisite; three had only a general software engineering (SE) course as a prerequisite; and the rest had no (SE or RE) prerequisites. Also, the recent IEEE/ACM curriculum for SE (Software Engineering, 2004) recommends general SE or software construction as prerequisites for SA courses, depending on the core package selected. Thus, our findings could be a trigger for possible streamlining of the prerequisites or for highlighting action that could possibly be taken either prior to, or during, the SA courses. Also, in the area of hiring and training, we have often seen in the software industry (in Canada at least) that architects' roles are not consciously assigned to agents with RKE. More often, the case is that these agents have more technical-oriented background (databases, networking, platforms, etc.). While this is important for architecting, it can leave a gap in the front-end and more conceptual areas of architecting, some of which have closer interaction with requirements. Also, for those architects without RKE, the findings from this study could indicate areas of improvement that can be used to structure training sessions for these agents.

Regardless, our study was conducted using the "mixed methods" approach (Creswell, 2003), i.e., there were both quantitative and qualitative aspects to this study. For example, the quantitative aspects included assessment of the quality of the software architectures developed by the participating groups; whereas, qualitative aspects included analysing interview transcripts for the kinds of feedback given to the participating groups. Architectural assessment instrument (included in the Appendix) and data-gathering templates were used to assess the impact of RKE on architectural quality. Also, semi-structured interviews (on architecting issues) were conducted and direct observations were made to subjectively assess the progress of the project during the process.

In total, there were fifteen projects, all of which used the Attribute Driven Design (ADD) method (Bass et al., 2003). Also, there were 60 participants, all final-year or graduate students. The domain to be architected was the banking domain, and there were some 85 high-level requirements to contend with. The primary task was to develop and document an architecture as per the guidelines and templates in (Bass et al., 2003). Additionally, there were many templates developed to capture the in-process rationale and partial work.

Another point to note is that our study was controlled while being exploratory. It was controlled in that there were two types of participants: with RKE and without RKE, as determined by background checks. It was exploratory in that, due to the lack of prior concrete knowledge on this topic, there was no tangible hypothesis on how the architects with RKE would compare – in technical terms -- against those without. Rather, hypotheses are expected to be an *outcome* of an exploratory study (Mason, 1996); this paper describes such a resultant hypothesis. Thus, an exploratory study is a foundational study for future studies on the subject matter.

Though the importance of conducting empirical studies in software engineering (SE) has been recognised (Tichy et al., 1995; Wieringa and Heerkens, 2006), Shaw's analysis (Shaw, 2003) of research papers submitted at a prominent 2002 SE conference suggests that only 12% were submitted in the category of "Design, evaluation, or analysis of a particular instance" and 0% in the category of "Feasibility study or exploration". Our own analysis of published papers, since the year 2000, in the fields[6] of RE and SA suggest that only 15% were in the above mentioned categories combined. This status of research suggests that studies such as

[6] We examined 552 papers from the Requirements Engineering Journal (years 2000 to 2007), IEEE Int. Requirements Engineering Conference (RE) (years 2000 to 2006), IEEE/IFIP Working International Conference on Software Architecture (WICSA) (years 2000 to 2007), the Software Requirements to Architectures Workshop (STRAW) (years 2001 and 2003), and the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ) (only year 2007, due to unavailability of online access,).

the one described in this paper are currently rather rare. Such sentiments were also voiced by many participants of the WICSA 2007 conference (WICSA 2007).

After describing related work in the next section, Section 3 describes the empirical study conducted; Section 4 describes the results and their interpretations; Section 5 discusses the implications of the findings; Section 6 describes possible future work; and Section 7 concludes this paper. Following the references, the Appendix contains the software architecture assessment instrument that was developed for this study.

2 Related Work

Historically, research work in the RE and SA fields have focused upon new technologies "within" these respective fields, e.g., requirements elicitation, analysis, prioritisation, methods and tools, or architecture design and evaluation methods. However, there is a growing body of work, albeit slowly, that is aimed at bridging these two areas. In this section, we examine such work and put it in the context of our research described in this paper.

Brandozzi and Perry's "Preskriptor" process (Brandozzi and Perry, 2003) is centred on an architectural "description" language and its associated process to systematically ensure that requirements are being satisfied. Egyed et al. in their CBSP (Component-Bus-System and Properties) method (Egyed et al., 2001) also use an intermediate language for expressing requirements in a form that more closely relates to architecture, where requirements are identified and categorized based on various properties such as whether they should be implemented as components, bus, system properties, and so on. Liu, W. and Easterbrook (Liu, W., and Easterbrook, 2003) extend this method by introducing a rule-based framework that allows for requirements-architecture mappings to be automated where possible. Liu, D. and Mei (Liu, D. and Mei, 2003), view the mapping from requirements to architecture in terms of features, essentially a higher-level abstraction of a set of relevant requirements as perceived by users (or customers).

Whereas these methods are primarily focused on formalizing the technical aspects of architecting, other researchers have proposed methods that are concerned more with human issues such as negotiation, real-world scenario forming, organisational culture, and risk assessment. In particular, In et al. (In et al., 2001) propose an eight-step framework that is based on existing RE and SA methods (WinWin and CBAM, respectively) to help stakeholders to elicit, negotiate, and evaluate requirements-architecture properties while concurrently executing these processes. Nord and Soni's work (Nord and Soni, 2003) deals with the identification and analysis of global factors - those that take into account more holistic issues such as the environment in which the system is built, developing organization, external technological solutions, flexibility or rigidity of requirements, and more. Their two-phase method is a means to analyse and resolve architectural issues introduced by global factors. Another method is Bass et al.'s stakeholder-centred Attribute-driven Design (ADD) (Bass et al., 2003), which focuses on iteratively building architectures based on the key architectural drivers of the system. These drivers are composed of key requirements and quality scenarios that shape the architecture. The drivers are input into the process where architectural patterns are created/selected to realize the tactics (i.e., the architectural design choices made), which in turn are aimed at satisfying the quality scenarios. Tradeoffs emerge in the patterns between various quality attributes, and the architects and other stakeholders must negotiate a resolution to these tradeoffs (similar in principle to the Architecture Tradeoff Analysis Method (ATAM) (Kazman et al., 2000) to finalize patterns that would represent an architecture that is most suited to meet the system's goals. Recently, a prototype tool, called ArchE (Bachmann et al., 2003b), has been developed to provide support to the ADD method. This support is in the form of modelling the functional responsibilities of the architecture, storing the quality scenarios, and through analysis of the architecture and quality scenarios, the tool suggests tactics that can be used to satisfy the quality requirements. To date, the tool supports modifiability and performance quality attributes.

A method that traces architectural concerns back to the requirements is the Architecture-centric Concern Analysis Method (ACCAM) (Wang et al., 2005). The method uses a Concern Traceability map (CT-map) that captures and presents architectural design decisions starting from software requirements through to the software architecture and these are then linked to architectural concerns that are identified in the architecture evaluation phase. Through a visual model, this method aids in identifying potentially problematic, or sensitive, requirements or decisions that resulted in the concerned architectural parts.

In (Schwanke 2005), Schwanke discusses the "Good Enough Architectural Requirements Process" (GEAR). This process is meant to further refine an initial set of requirements through architectural means. The process is based on three architectural requirements engineering approaches: model-driven requirements engineering (where elicited candidate-requirements are modelled as use cases, activity diagrams, state charts, etc.), quality attribute scenarios (used to elicit, document and prioritize stakeholder concerns), and global analysis (a general way of organizing information about the problem context that surrounds the architecture). The main purpose of the process is to show where the above approaches overlap and where they complement each other, providing insight into the identification of architectural requirements.

Poort et al. (Poort et al., 2004) propose a framework for mapping non-functional requirements onto functional requirements for architectural design. Their framework is based on a model of the relationship between requirements and architecture, and a repeatable method that can transform requirements into system design, and generates a "risk-list" based on conflicting requirements. Their framework is not meant to provide a means for achieving specific quality attributes; it is used to highlight the relationships between the requirements, their conflicts and architectural means of resolving them.

Other work is that by Rapanotti et al. (Rapanotti et al., 2004) where the concept of problem frames is extended into "architecture frames" which capture information about architectural styles and their interaction with the problem space. The benefit of this mechanism is that in introducing solution-oriented approaches early in development, one can refine problem analysis.

In (Damian and Chisan, 2006), Damian and Chisan report on a large-scale case study on the effectiveness of requirements engineering processes on other development processes such as architecting, lower-level design and implementation.

Also, they link many problems that occur later in development back to problems that originated during the requirements phase. However, the sorts of problems they have investigated are quite complementary to the ones we have investigated in our study, For example, whereas, we have investigated issues such as impact of RKE on architecting tasks such as: quality drivers determination, selection or determination of tactics, integration of tactics into architectural patterns, they have investigated issues such as requirements not being properly documented and shared, relying on word-of-mouth, incompleteness and inconsistencies in requirements, etc.

In (Ferrari and Madhavji, 2006), Ferrari and Madhavji report on a multiple-case study that investigated requirements-oriented problems that are encountered while architecting. Overall, they found that approximately 35% of the problems encountered during architecting were requirements-oriented. Also, specific problem areas together with their severity were identified (such as, quality satisfaction, requirements understanding and quality drivers) as well as the relative frequency of problems occurring in these areas. Implications of this work are on improving methods, tools, and techniques to transition from requirements to architecture.

In another study (Miller and Madhavji, 2007), Miller and Madhavji investigate the interaction between requirements and systems architectures. Specifically, they explore the effect of systems architecture has on the decisions that are made during requirements elicitation of an evolving system. They identify nine architectural aspects (e.g., existing hardware, reusability of modules, and architectural patterns) that can have an effect on new requirements decisions, as well as three principal ways in which a previous architecture can affect evolving requirements work, i.e., as an enabler, as a constraint and as an influence, apart from the null case.

Progress in the area of empirical studies on requirements-architecture interplay, however, is still rather slow. In part, opportunities to conduct industrial-scale case studies are quite rare, and almost negligible, if not impossible, when multiple teams are considered. Empirical studies conducted in a learning environment are a next possibility. Previous studies in the areas of software inspections (Thelin, 2004), RE (Easterbrook et al., 2005; Berander, 2004), and Lead-time impact assessment (Hoest et al., 2000), and critical analysis of using students as subjects (Carver et al., 2003) to name a few, have shown the benefits of conducting empirical studies in a learning environment. The study described in this paper is another of such studies but in the area of transitioning from RE to SA.

3 The Empirical Study

We describe here the study that was conducted to explore the impact of requirements knowledge and experience (RKE) on software architecting. The sub-sections deal with: the research questions, study variables, study design, participants involved, research procedures, the requirements document, the architecting project, and threats to validity. The sub-section on threats to validity concludes this section since it discusses the threats that may exist in the content of the sub-sections that precede it.

3.1 Research Questions

Our overarching research question is:

RQ1: While architecting a software system, how do the architects with software requirements knowledge and experience compare against those without such knowledge and experience?

This question deals with the relative performance of two different groups (RKE and non-RKE) when creating a systems architecture. In particular, we are interested in the difference in the overall architectural product quality between the two different types of groups. The definition of architectural quality, and how it is operationalized in this study is described in Section 3.2.

There are two noteworthy points concerning the research question. One is the participants' background. We had analysed the background knowledge and skills of the participants and, as described later, the most critical aspect was their separation in terms of RKE vs. non-RKE background. Another point is the research hypothesis. While it is obvious that requirements and architecture are inter-related, the lack of previous human-centred studies on this topic meant that we had no tangible hypothesis on how the architects with RKE would compare – in technical terms -- against those without. This is why our study was an *exploratory* one, which, as it turns out, gave rise to a hypothesis (described later) during the conduct of the study.

This quantitatively-driven question above is complemented by the following qualitative question, RQ2, which seeks the underlying reasons for the findings of RQ1.

RQ2: What could be the underlying reasons for the performance of the respective groups?

We define the performance of the respective groups to be their relative abilities to produce a better overall architecture. In the investigation of this question, we will be examining in-process artefacts, partial products, and the final architecture produced.

The above listed questions deal with several dimensions of interest: (a) the process of architecting, (b) the product architecture, and (c) the requirements background. Detailed findings centered on these issues should throw some light on the relative difference between those architects with RKE and those without.

3.2 Study Variables

In this subsection, the variables of interest are introduced and discussed, along with their associated metrics (see Table 2-1).

Variable	**Type of Variable**	**Metric**
RKE	Independent	A categorical variable, either the participant has RKE or not.
Architecture Quality	Dependent	Evaluation of final architectural quality based on a set of criteria (measured through an instrument discussed in section 3.7.1.2, and included in the Appendix). Examples of areas that are evaluated include module decomposition of the system, behaviour models, and interface specification.
Effort	Extraneous	Time in hours expended on the project
Feedback	Extraneous	Number of feedback interactions between researchers and participants
Academic Background	Extraneous	Average of all marks from courses participants have taken at the University level

Table 2-1. Summary of study variables

The dependent variable of the study is *Architecture Quality*, defined by the final architectural quality based on a set of evaluation criteria (as defined in Section 3.8.1, some examples include assessing the quality of the module decomposition structure, component and connector view, and interface specification).

The treatment, or independent, variable is *RE knowledge or experience* (RKE), which is defined as the subjects having previously taken a RE course. In this course, students were taught generic Requirements knowledge, where they learnt such topics as elicitation, modelling, analysis of requirements, negotiation, prioritisation, quality drivers, viewpoints, specification, validation, traceability, process, management, etc. Further details can be found in (Kotonya and Sommerville, 1998). The treatment of RKE was not administered in *this* study but is innate to the subjects based on their previous experience. The treatment variable is *controlled* in that it dictates the type of study group (RKE vs. non-RKE) a subject belongs to.

In addition, there are three *extraneous* (or *factor)* variables that must be accounted for in the study. As discussed in later sub-sections, we needed to measure and statistically eliminate any possible *extraneous* variables because they may have an impact on Architecture Quality (the dependent variable). These *extraneous* variables are: *Academic Background*, defined by the average marks (out of 100) obtained by the subjects in previously taken courses; *Effort* expended by the subjects as measured in hours spent architecting the system; and *Feedback*, defined by the type and amount of external assistance (called feedback) sought by the subjects to complete the projects.

3.3 Experiment Design

The type of this study was a *quasi-experimental design*, where the researcher uses control and experimental groups, and randomly assigns the participants *within the different groups* (Creswell, 2003). A *quasi-experimental design*[7] is used when

[7] We emphasize this point because we feel that in Software Engineering this type of studies are not prevalent as yet.

the main property of interest (in our case, RKE vs. non-RKE) is *innate* to the participants *before* the study is conducted[8].

In this category of design, our study fits the *Posttest-Only Control-Group Design* (Campbell and Stanley, 1963; Sheskin 2004). This type of experiment investigates and compares the effects of a treatment on two or more groups; in our case two: Group A being the "treatment" (RKE) group (meaning that Group A got the RKE training as described in the previous section), and Group B being the "control" (non-RKE) group, as depicted below:

```
Group A:     X R----------------- O
Group B:       R----------------- O
```

where: O represents observation during architecting (along with the results from the other independent variables -- Academic Background, Effort and Feedback); X represents the treatment (RKE); R represents random assignment *within* group; and the left-to-right dimension indicates the temporal order of the procedures in the experiment.

3.4 Participants

We used availability (or *convenience*) sampling (Creswell 2003), where the participants were drawn from the final year *Software Architecture* course at the University of Western Ontario. There were fifteen architecting teams, each comprised of four members. It turned out that approximately 40% of the subjects had RKE background; whereas, the rest did not – based on a background survey and academic records, so there were seven RKE teams and eight non-RKE teams. Membership of each team was determined through random selection by an independent person.

[8] We realize that, in addition to the background questionnaire, we could have assessed the subjects' knowledge on RE. The results of this assessment could have been used to validate the results from the background questionnaire. However, due to pedagogical factors (the subjects were enrolled in an architecting course), we could not issue an assessment only on requirements. However, we did not have any doubts in the categorisation of the subjects based on their background data.

3.5 The Requirements Document

The system to be architected was in the "banking" domain. The application included three different modes of banking for the clients: ATM, internet banking and telephone banking services; access and reporting features for the banking staff; client and financial database; various quality drivers, such as security, availability, performance, usability, maintainability, and others. In all, there were some eighty-five high-level requirements to contend with, which is quite sizeable for a term project.

A project in the banking domain was used since the banking domain is sophisticated enough to provide the basis for a substantial architecture. Conversely, typically people are familiar with banking (although not necessarily with its design) which would minimize the possibility of domain-complexity interfering with system architecting.

The requirements document for the system was obtained from an external source. The requirements process followed (i.e., elicitation, analysis, validation, prioritization, and documentation) is described in (Kotonya and Sommerville, 1998). Prior to conducting the study, these requirements were re-validated by a team of five experts for acceptability in general, "and for any serious or obvious flaws"; the semantic-content of the document was otherwise not altered. We did this in order to reduce *researcher bias* in the study. We also did not want to "fix" the document to the point where it was considered "perfect". In a real world setting, the requirements documents given for architecting or system development are not always "perfect", and we wanted to emulate this by delivering an acceptable document to the participants. Also, none of the participants of this study (i.e., the architecting teams) had any involvement in the requirements forming process.

3.6 The Architecting Project

Given these requirements, each of the fifteen teams independently developed an architecture using the ADD method (Bass et al., 2003). The key steps of this method include: iteratively decomposing a selected module, choosing architectural drivers from the scenarios and functional requirements, choosing or creating an architectural pattern (using appropriate tactics) that satisfies the architectural drivers, identifying child modules to implement the tactics, instantiating the modules with functionality, defining interfaces, verifying and refining use cases and quality scenarios and making them constraints for the child modules.

Each team had to develop and document the system architecture and, in the process, capture in the defined templates such items as: design decisions, rationale, underlying assumptions, issues arising, resolution of items, etc. In addition, each team had the freedom to seek help (called "feedback" below) on any difficulties they faced during their project.

3.7 Research Procedures

In this section, we describe the research procedures used to conduct the study. First, we discuss methods for data collection from the architecting projects (both for gathering project data from the subjects as well as for gathering architectural quality data through the use of an instrument), followed by procedures for analysing feedback data.

3.7.1. Data Collection and Instrument Design

3.7.1.1. Project Data Collection:

There were several key sources of qualitative and quantitative project information for any given team: intra-team email communications, data templates, partial products, feedback sessions, and the final architecture. The data templates documented such items as: the decisions made while architecting the system, alternatives, underlying assumptions, rationale, issues, resolution, work breakdown structure, meeting minutes, and time-logs.

The qualitative data was dominated by emails as well as feedback sessions that constituted approximately 50 hours of recorded interactive sessions, which were subsequently transcribed by three domain experts and verified for accuracy. Each session was at least an hour long, involving one team and the project staff. The data templates and partial products were among the fodder for raising issues during the feedback sessions (see Table 2-2 for more details).

Template Name	User	Purpose and Summary of Instrument
Time Log Template (TLT)	Participants	The participants filled the time spent on any project related activity in this form on an ongoing basis. The effort metric is directly related to this template.
Decisions, Issues, Rationale Template (DIRT)	Participants	Each team had a team DIRT and each individual member of the team had their own DIRT. The DIRT was used so that participants could enter more qualitative data: all their design decisions, project issues and rationale relating to the project. They filled this document on an ongoing basis during the project.
Architectural Assessment Instrument	Assessors	This instrument is used by the assessors to measure the final quality of the participants' architectures. This instrument is discussed more in detail below and is included in the Appendix.

Table 2-2. Data collection templates

The feedback session data was gathered using qualitative techniques that are more commonly associated with Social Sciences (Creswell 2003). For example, ethnographic methods (Hall 2004) were used such as participant observation and semi-structured interviews, and textual document analysis. The lead researcher attended all these sessions but, to ensure high quality of the feedback, a second researcher also attended every meeting.

The variety of information sources and numerous feedback sessions helped to obtain rich data concerning any topic or theme that arose within the research domain investigated. They also allowed the staff to monitor and inspect data for consistency and completeness and deal with problems efficiently and effectively.

3.7.1.2. Architecture Quality Assessment:

Besides the described qualitative data, quantitative data was also gathered from the assessment of the final architectures. For this purpose, we developed and validated an architectural assessment instrument (see the Appendix). This was used by five experienced software engineers (with experience ranging from 5 to 27 years in SE and research) to assess the resultant architectures from the study projects.

In short, the instrument uses a mix of scale types; mostly continuous 7-point Likert-scale, but also categorical scales. The Likert-scale is used to rate specific aspects of architectural quality. This scale is typically used in surveys or instruments that are used by external assessors of a program or artefact. They ask for a level of agreement about a specific construct, and although the scales are ordinal, they can generally be used as interval data in statistical tests. The 7-point scale is typically used when the data requires a fine level of granularity but still manageable for the users of the scale (De Vaus, 2002).

The quality-criteria for the instrument items are derived from the project documentation guide, SA literature, and the standard templates from (Bass et al., 2003). The central components measured by the instrument include: Modelling the environment; Use Cases; Quality Scenarios; Module Decomposition structure; Component and Connector structure; Deployment structure; Interface specification; Modelling the dynamic behaviour of the system; Overall Architectural properties; Architectural reasoning; View descriptions; and Overall documentation quality. More details can be found in the Appendix.

Also, to ensure *content* and *face* validity (Carmines and Zeller, 1991) of the instrument, there were numerous iterations and stages in the design and implementation of the instrument. This included reviews and establishing relative weights for different items corresponding to the project requirements, and had intimately involved six knowledgeable software engineers with RE and SA experience. The instrument was subsequently piloted by two raters on several documented architectures prior to its use for quality assessment of all the architectures.

3.7.2. Feedback Data Procedures

To assess the issues that arose in the feedback sessions (where subjects could freely interact with staff), we analyzed the transcribed data and emails. In essence, we counted the frequency of the various types of feedback (i.e., severity of feedback, and kind of technical activity). The technical activities were identified *a priori* from the ADD method (Bass et al., 2003) and relevant research literature and were validated by six knowledgeable software engineers over several iterations. This resulted in over 20 categories, to name a few: *Requirements Understanding*, *Context*

Modelling, Quality Drivers Determination, Interface Specification, Behaviour Modeling, and *Architectural Reasoning.*

To determine the *severity* of feedback items, *thematic coding*[9] was conducted on the transcribed data set, using the identified categories and discovering any new *inductive* categories while coding (Mason 1996). QSR's NUD*IST[10] 4.0 was used for *thematic coding*; it manages and stores all the emerging codes, and allows easy retrieval of text units that have been coded.

This analysis resulted in the identification of three severity levels of feedback given to the various teams: "*Hint*" (Light), "*Explanation*" (Medium) and "*Give Aways*" (Heavy). Below, we give examples of this where: ***P*** represents a participant and ***S*** the teaching staff:

- "*Hint*" – Participants only needed a "hint" to proceed with their architectural design. Example: ***P****: Do our concrete scenarios help us in shaping our architectural patterns?* ***S****: Yes, but not directly so. Concrete scenarios are there to identify things that will happen often in the system, and with those identified, you can prioritize what is important in the system.*
- "*Explanation*" – Participants needed a detailed *explanation* in order to proceed further in their solution design. Example: ***P****: We weren't sure how specific we should be getting with the quality scenarios. Here's a few that deal with availability, but some seem to be specific, like this one about a power failure, which will not happen often. We're just confused with these scenarios.* ***S****: So this is something that would happen a lot, or could be a scenario that has a particular high impact on the system, so that's the purpose of the quality scenarios, you're just thinking of ... obviously there's hundreds of scenarios you can think of in any given system, but, you're trying to think of those that are most important in the system, e.g., the ones that will be encountered the most often.*

[9] *Thematic coding* is a qualitative data analysis procedure where the researcher develops categories of concepts and themes that emerge from the data source. It is an 'open' process in that the researcher makes no prior assumptions about what the findings may be.
[10] QSR NUD*IST 4.0. QSR International Pty Ltd., 1999. Available at http://www.qsrinternational.com.

- "*Give Away*" – Participants needed a full solution to a particular problem for them to move forward in their design. Example: ***P**: we kind of just did the drivers as the requirements dictated, the ones that seemed the most important. Is that the way it's supposed to happen? I was a little worried about that, so do you want say performance, and then just say why you do based on the requirements? **S**: ummm, yeah, you could use the requirements as an example of why you consider performance a key driver. I mean that's where quality scenarios are critical. Scenarios deal with quality issues, nonfunctional, and so really the scenarios, I mean you could come up with hundreds of them ... but you're trying to come up with the ones that seem the most important based on the requirements and your own banking knowledge. And then, you can use these scenarios and say ok, so this will happen a lot, or a failure of this scenario would be devastating in our context, so it is high priority. Once this is done, you come up with tactics to resolve those scenarios. But I mean a scenario is attached to a quality attribute. So really, it's just a way of prioritizing, your key quality attributes.*

This analysis procedure was then validated through piloting, independent review and re-coding when error rate exceeded 20%.

3.8 Threats to Validity

We classify threats into those *internal* and those *external* to the project, conclusion and construct validity, as well as qualitative study threats. We focus here only on those threats we identified as applying to our study. Description of other types of threats can be found in (Wohlin 2000).

3.8.1 Internal Validity

Internal validity is the degree to in which the independent variable was responsible for the change in the dependent variable, and not because of other confounding variables (Carmines and Zeller 1991). There are numerous types of internal validity threats that are identified in the literature (Campbell and Stanley, 1963) which are discussed below:

Maturation – This is present when a physical or mental change occurs during the study and it affects the participants' performance on the dependant variable (e.g., Architecting). With weekly motivation and feedback meetings, and random assessments of the various templates, we did not notice *maturation* within or across the study groups.

Instrumentation – This refers to any change that occurs in the way that a dependant variable is measured in the study. There was no change in the definition of the dependent variable during the study. Also, the evaluators were quite familiar with the assessment instrument. Moreover, the assessments were reviewed by others to identify anomalies.

Selection – This refers to selecting participants for the two groups that have different characteristics. In our study, we were most concerned with the possibility of one group having "brighter" students than the other. However, the past academic data shows that the two groups were almost identical in this respect. Other issues that here are whether one group simply has more experience than the other (e.g., through industry or other experience). Based on our background investigation and interviews there were no such cases.

The other *selection* threat that could have existed is the discrepancy in the number of courses and types of courses that were taken by the students (e.g., the RKE group taking more SE-oriented courses than the non-RKE group, thereby giving the former possible advantage in the Architecting area over the latter group). This threat is mitigated by the current course curriculum at UWO where by the time the students reach the 4th year the SE courses taken are roughly the same for all the students. This is supported by the background academic records (which we obtained from the University office) and background data that we collected from all the subjects.

Researcher Bias – This occurs when the researcher, knowingly or unknowingly, influences the outcome of the study. To mitigate this threat, multiple researchers and domain experts were used in the study processes (e.g., research design, research objective validation, data collection and gathering, validation of data, instrument and template design, data analysis, and interpretation of the results). These people had no conflict of interest in the study and therefore we consider them not to have any bias toward the study results.

Hawthorne Effect – This threat is when the mere presence of researchers watching the participants causes a change in their performance. This threat does not typically exist in studies involving two or more group studies, as the observation of the groups could lead to increased performance by all study groups, but the *difference* is more or less equal.

3.8.2. External Validity

External validity is the degree to which any findings from the study can be "generalized to and across populations of persons, settings, and time." (Creswell 2003) There are three types of validity that apply to *external validity*: *Population, ecological* and *temporal*. Each of these and how they possibly could apply to our study are now discussed.

Population validity – This refers to the *generalization* of the sample to the population, and the sample results to the different types of people within the population. This is a risk in our study and it arises from using students as the study participants. This threat is directly imposed on the *generalizability* of any findings for application in *industrial* contexts. However, these results would likely be generalizable to relatively new workers in industry, as their experience level is comparable to that of the participants in this study. Also note that there are strong implications of the findings on systems architecture *training*, and pedagogy. It is important here to separate "development" from "training" because they are complementary activities in industry. That said, recent research in Software Engineering (Hoest et al, 2000; Runeson 2000; Thelin 2004; Easterbrook et al.; 2005; Berander, 2004; Carver et al., 2003), has shown positive experiences when conducting empirical studies involving students in a learning environment.

Ecological validity – This threat refers to the generalizibility of the study results across all settings. As with *population validity,* the academic setting can be quite different from an industrial context so the threat is present. However, the project was loosely structured (as opposed to a strict "laboratory" setting) so that the setting could more mirror real-world work.

Temporal validity - This is present when the results of a study can be *generalized* across time. In our study it is difficult to discern whether this holds, since there are no results to compare to, as this is the first study of its kind. From an intuitive perspective, there is no reason to believe that our study does not maintain *temporal validity.*

3.8.3. Qualitative Validity

Since our study was a mixed-method approach (Creswell 2003) involving both quantitative and qualitative study techniques, we discuss here possible threats to the qualitative aspects of our study.

In qualitative studies, a validation technique, called *triangulation* (Berg, 2007), is used to ensure validity in the study. *Triangulation* is a method of establishing the accuracy of a study's findings by comparing three or more types of independent points of view on a given aspect of the research process (methodology, data, etc.) (Berg, 2007) There are different types of *triangulation* that can be used together to form a strong basis of validity. In this section, we will discuss how we used three different types of *triangulation* to ensure validity in our study. The *triangulations* used were: *data triangulation, methodological triangulation,* and *investigator triangulation.*

3.8.3.1 Data Triangulation

Data triangulation is the use of different sources of data/information on which the study results are based. If there is consistency in the data/information provided across the various data sources that are used, then this suggests that the data is valid. In our study, as mentioned in section 3.7.1 (Data Collection and Instrument Design), our data-set came from numerous sources including the feedback sessions, intra-team e-mail communications, and various data collection templates that the participants had to complete.

3.8.3.2 Methodological Triangulation

Methodological triangulation is the use of different methodological techniques (that could be either quantitative or qualitative) in the study and, if the conclusions from each method are consistent, then validity is increased. In our study, we used various qualitative and quantitative methods such as participant observation, semi-structured interviews, document analysis, quantitative content analysis and statistical analysis on the final architecture quality. The resultant data from these various methods and its subsequent analysis (see Section 4) showed similar conclusions. This consistency establishes methodological validity in our study.

3.8.3.3 Investigator Triangulation

Investigator triangulation is the use of different researchers in the various study processes, and then if there is concurrence between the outputs of these methods used by the researchers, then investigator triangulation is established. In this study, multiple researchers were used for the feedback sessions, assessing the final architectures, and the analysis and interpretation of the feedback data. In each of these processes, there was agreement among the different researchers. For example, during the feedback sessions, there were open discussions with the subjects involving different researchers and on each occasion there was consensus. Likewise, the coding of the transcribed data (see Section 3.7.2) was analysed by several raters and when there was a divergence of 20% or more, the coding was redone to a satisfactory conclusion. Similarly, two raters conducted architecture assessments, and if there was substantial divergence, a third researcher was brought in to assess the situation and establish an agreement on the rating.

3.8.4 Construct and Conclusion Validity

Construct validity is the degree to which inferences can be made from the measures in the study to the theoretical constructs on which those measures were based. In our study, the dependent variable *architecture quality* was measured through the use of an architectural assessment instrument. In section 3.7.1 we discuss the construct validity threats (more specifically, *content* and *face* validity) and how we mitigated these threats with respect to the instrument that was created to measure the dependent variable. Also, the three levels of feedback (Hint, Explanation and Give Away) were defined inductively and concurred upon by several experts prior to the use of these categories across the transcribed data.

Conclusion validity is the degree to which conclusions we make based on our data are reasonable. We discuss the results in the next section where we also demonstrate that our study did not violate conclusion validity.

4 *Results and Interpretations*

Now, we describe the results and their interpretations concerning the two main research questions (RQ1 and RQ2 – see section 3.1).

4.1 Architects with RE knowledge and experience vs. those without (RQ1)

Here, we are interested in determining the relative performances between those architects with RE knowledge and those without. In order to do this, we will quantitatively analyze using statistical techniques the relative difference between the two types of groups with respect to the final architectures they submitted. Prior to delving into the details of the analysis, we discuss the emergence of a hypothesis on which the statistical testing is based.

Typically, in exploratory studies, research hypotheses are not stated from the outset. Rather, they are generated from the results of the study or during the execution of the study (Mason, 1996). In our study, as we attended the work sessions of the architecting teams (feedback sessions) and reviewed early architectural artifacts emerging from the process, it seemed that the RE knowledgeable teams were performing better in terms of the quality. Thus, the following hypothesis emerged:

H1: Architects with RKE develop better quality systems architectures than do architects without RKE.

Before conducting a statistical test, we set the alpha level (or level of *significance)* to be .05.

This hypothesis was analyzed and tested using analysis of covariance (ANCOVA[11]) with three extraneous variables (the covariates, as identified and discussed in section 3.2):

(a) the *time spent on the project*,

(b) the *frequency of the aggregate of the three different levels*[12] *of feedback* (see section 3.4.2), and

(c) *the academic background,*

and a dependent variable: *architecture quality*. This was done to statistically factor out these extraneous variables and their impact on architecture quality to determine the value of the dependent variable without any "interference" from other variables. As discussed in section 3.3, our study was a *quasi-experiment* (where subjects with RKE and non-RKE were naturally divided prior to the start of this study) and therefore random sampling was not possible to negate the impact of these other factors. **Table 2-3** presents the *means (as well as other descriptive statistics)* of the architecture quality of the two groups (RKE and non-RKE).

In **Table 2-3**, we see that the RKE groups received a mean of 78.5 on the architecture quality variable, against 62.4 for the non-RKE groups. We conducted a one-way tail t-test on the means and it resulted in p= .076 or a 92.4% confidence rating which is not significant. Based on the actual averages, it would intuitively seem that the difference is substantial, however, the standard deviation reports widely distributed values within groups (16.5 for RKE, 15.7 for non-RKE) which suggests that other factors had an influence on the architecture quality values. Therefore, the ANCOVA test had to be used to statistically test the data from Table 3, in order to eliminate the effect of the confounding factors: academic background, feedback and

[11] ANCOVA (Wildt and Ahtola, 1978) is an extension to ANOVA and is used to statistically control extraneous variables when experimental control cannot be used. It is used to reduce experimental error or to remove the effects of extraneous variables. ANCOVA is based on linear prediction or regression of the covariates; using prediction equations to predict the values of the dependent variable on the basis of the values of the covariates, after which these predicted scores and means are subtracted from the corresponding values of the dependent variable.

[12] We performed the analysis on the three separate levels of feedback (see section 3.7.2) but they did not show any significant impact on the dependent variable. For simplicity of the model, we discuss the feedback variable in this sub-section as a single aggregate variable.

effort. Without such further analysis, a statistically significant result cannot be determined.

	Descriptive Statistics	Arch. Quality	Academic Background	Feedback	Effort
Non-RKE	Mean	62.4	72.9	65.4	139.0
	Std. Dev.	16.5	4.0	44.0	34.5
RKE	Mean	78.5	73.9	51.3	139.8
	Std. Dev.	15.7	4.4	30.1	41.0
Total	Mean	69.9	73.4	58.8	139.4
	Std. Dev.	17.6	4.1	37.5	36.3

Table 2-3. Descriptive statistics for study variables

Looking at the other variables of interest, in their respective academic backgrounds there was a one-point difference between them (72.9 for non-RKE, 73.9 for RKE), this, which is not a significant difference so the only possible difference in the two types of groups with respect to Architectural Quality could be due to the extraneous variables Feedback and Effort. In looking at their descriptive statistics, the distribution seems to be quite varied (for Feedback, the standard deviation was 44.0 for non-RKE and 30.1 for RKE), suggesting that the amount of feedback sought was quite different among the different teams between both types of groups. The effort expended by both types of groups was almost identical (RKE: 139.8, non-RKE: 139.0), suggesting that the higher mean for the RKE groups was not due to any increased effort they expended on the project. We now discuss the statistical test used to determine the statistically significant values of the various variables and their impact on the architecture quality variable.

The ANCOVA test was performed using SPSS on the architectural quality variable as the dependent variable (see Table 2-1), the results of which are shown in Table 2-4.

Source of Variation	DF	Sum of Squares	Mean Square	F	Sig.
Academic Background	1	1848.238	1848.238	15.431	.003
Feedback	1	118.667	118.667	.991	.343
Effort	1	35.095	35.095	.293	.600
RKE	1	713.902	713.902	5.96	.035
Error	10	1197.727	119.773	▇	▇
Total	15	77590.815	▇	▇	▇

Table 2-4. ANCOVA table for architecture quality

In Table 2-4, we see that there is a statistically significant difference between the two groups (RKE and non-RKE – see the variable "RKE") at p=0.035 or a 96.5% confidence interval. The strongest predictor of high *architecture quality*, however, was the participants' Academic Background (significant at p=0.003 or a 99.7% confidence interval). The findings that academic background was the strongest predictor is not entirely surprising; it is commonly known that highly knowledgeable software developers will more likely produce high-quality software artefacts (Boehm, 2002). However, what is surprising is the *extent* of the impact of RKE on architecting (approximately 16% difference between the two types of groups), especially considering the academic background of both types of groups was approximately equal. We intuitively expected that there could be some impact, especially early in the process where requirements are intensely dealt with, but as we will see in the next sub-section, the difference permeated throughout the entire architecting process.

The Feedback and Effort variables, despite initially thinking that they would have some impact on the quality of the architectures produced, did not have a significant impact on the resultant *architecture quality.*[13] It is not that the Feedback

[13] In fact, the feedback and effort variables did not make "good" covariates, in that the relationship to the dependent variable was not strictly linear in our case, which is one of the underlying assumptions about the data that ANCOVA requires. Note that having more variables in an ANCOVA model can reduce the statistical power of the test, however, because the two variables and the statistical model

and Effort had no impact for every group; for some groups that had relatively low academic background scores and non-RKE but still produced an average quality architecture, it is mostly attributable to the fact that they put in more effort and received more feedback than other groups. However, amount of Effort and Feedback was substantially less for teams that had reasonable academic background scores and RKE, again showing that the type of experience (RKE) and Academic Background were the prominent indicators of higher architectural product quality.

Our results from the ANCOVA test show substantial support for H1 (see above). Though we recommend that these results should be interpreted with caution because ours was an *exploratory* study, these results can be a strong motivator for future more tightly controlled *true experiments.*

4.2 Possible underlying reasons for the relative performance of the respective groups (RQ2)

In order to determine the possible underlying reasons for the performance of the RKE and non-RKE groups, we can perform two primary analyses: (i) of the findings from the feedback severity levels (see section 3.7.2) and (ii) of the final architectures developed by the teams in the two types of groups. Frequency counts and bar charts will be used to highlight the feedback and architectural product data in the fine-grained categories (see section 3.7.2 for feedback and section 3.7.1.2 for final architecture quality). The feedback and final architecture data sources are used because they provide complementary views on in-process and end product work by the various groups. It is important to note that statistical analysis is *not* done on the fine-grained categories because they were *inductive* and emerged within the context of our study[14]. Also, the feedback variance was high (see section 4.1) among the groups because this aspect of the study was not meant to be controlled. The groups themselves could seek as much feedback as they required. These next sub-sections

still showed significant values they did not diminish the model in any way.

[14] Analysing the feedback transcripts discovered the fine-grained categories; they were not *apriori* identified. Because of this, it is difficult to statistically test this data since the analysis procedure was *exploratory* and there could be other data types that could possibly have been in the transcript but might not have been discovered.

are meant to *support* and *reason* about the overall difference in the two groups' performance.

4.2.1 Analysis of Feedback

Recall that in section 3.7.2 we categorised feedback into: Hints (i.e., light), Explanations (i.e., medium) and Give Away's (i.e., heavy). Table 2-5 shows that the average amount of feedback given per team was quite similar for both RKE and non-RKE groups in the categories of Hints and Give Aways, but there is clearly a difference in the average number of Explanations given where the non-RKE teams received on average almost seven more explanations.

Group	**Average # of "Hints"**	**Average # of "Explanations"**	**Average # of "Give Aways"**
RKE	23	20.7	7.6
Non-RKE	25.9	28	7.7

Table 2-5. Average feedback per team

The three bar charts (Figures 2-1-3) that follow show, respectively, the difference in the three levels of feedback for the RKE and non-RKE groups against the technical activities of the entire architecting process that were derived from the *a priori* categories established in section 3.8.2. Note that for the technical areas not listed in the charts, there was not much of a difference between the RKE and non-RKE groups.

4.2.1.1 Hints

Figure 2-1 highlights a few striking differences between the two groups: *Quality scenarios, Tactics, Quality satisfaction and Pattern determination* are all 1-2 hints higher per team in the non-RKE group, suggesting that they needed more help in these areas.

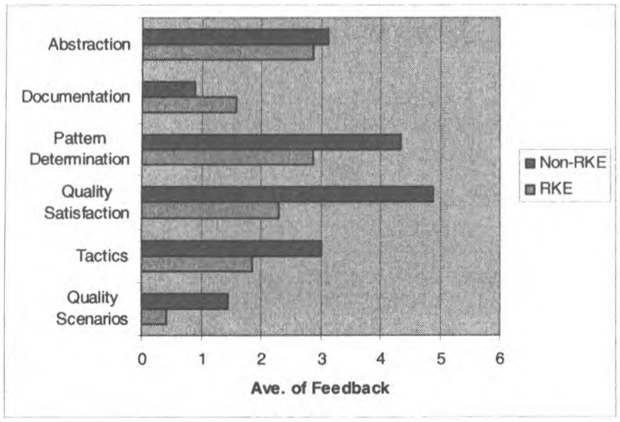


Figure 2-1. Difference of "hints"

This finding gives substance to Bachmann et al.'s intuition (Bachmann et al., 2003a) that the link between RE and SA is in the area of quality and tactics.

An area that we see that the non-RKE group did marginally better than the RKE group was in *documentation.* We identify this minor difference here so as to assess whether this forms a trend in Explanations and Give Aways.

4.2.1.2 Explanations

Figure 2-2 shows that in most tasks the non-RKE group received slightly more "Explanation" feedback than the RKE group. *Pattern Determination, Tactics,* and *Quality satisfaction* continue to be the categories with the biggest differences, but here, the major difference is with *tactics*, where the non-RKE teams received approximately two more explanations than the RKE group. *Tactics* are design decisions concerning the satisfaction of quality issues. These quality issues are introduced in the requirements phase, and are therefore quite requirements oriented.

A few other categories also seem to favour the RKE group (in terms of less feedback): *allocating functionality* and *abstraction. Allocating functionality* occurs after a pattern has been determined, and is where the functionality, as depicted in use cases or functional requirements, is allocated to the appropriate components.

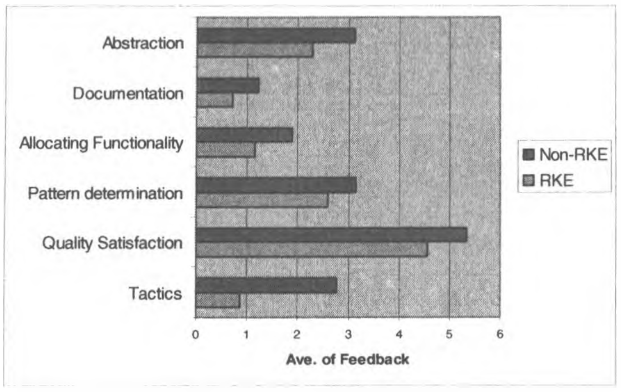


Figure 2-2. Difference of "explanations"

This activity clearly requires an understanding of: (i) the requirements, (specifically in knowing how the functional requirements are constrained and supported by the various quality drivers), and (ii) how these requirements are associated with their accompanying component arrangement (pattern).

The category *abstraction* represents the degree to which the architecture is too detailed or too abstract. This does not provide any *explicit* links to RE, but instead could possibly involve a "frame of mind" (i.e., an *implicit* link) as seems to be suggested by the following interaction:

<u>**RE**</u> (**P1** and **P2** are participants, **S** is staff)

P1: *I can't even imagine how the groups are doing that don't have requirements knowledge.*

S: Do you find that having done requirements is actually useful?

P1: *oh yeah*

S: in which way?

P1: *Because I just don't have to think about things as much, I can look at the requirements and immediately get sense of how things will work, I don't know, I'm just used to working with them, so...*

P2: *I know that at the end of this course just like with requirements, I'm going to be always looking and thinking about the architecture. Like now when I look at any projects, requirements are key on my mind. Before I didn't even think about them, like for programming and, well now I feel they are extremely important, and that this architecture is going to be in my mind* *as well... I know it would have helped me a lot to be able to think about other projects before like I can now...*

The marginal *hint* difference in the category of *documentation (*as described in 4.2.1.1) in favour of the non-RKE group is not repeated for Explanations. This suggests that there is no substantial difference between the two groups in this category.

4.2.1.3 Give Aways

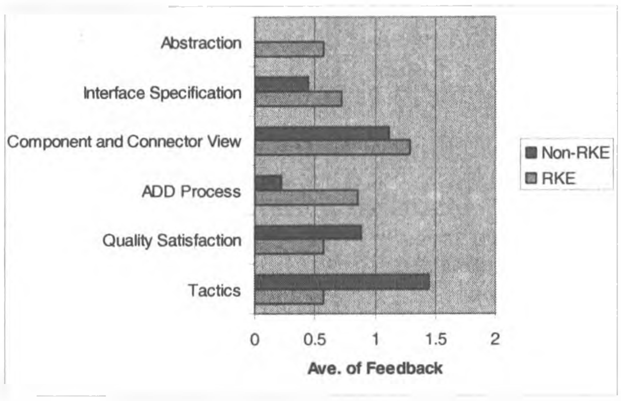


Figure 2-3. Difference of "give aways"

Figure 2-3 shows mostly the same trends continuing, although, it is difficult to assess since the number of data points is much lower than in the hints and explanations. The RKE group again required less feedback in *quality satisfaction* and *tactics.* A new category that emerged as one that required more relative feedback for both groups was the *component and connector* view. The difference between the two groups, however, is marginal.

In the Explanations section (see 4.2.1.2 above), we discussed *abstraction* and gave an interaction-example that demonstrates why the RKE group possibly does better in this area. However, in the Give Aways, there is clearly a difference in the two groups in the opposite direction; the RKE group sought in excess of half a "give away" more than the non-RKE group. When examining the details of the feedback given, the entire contribution of *abstraction* feedback came from only one team in the

RKE group, suggesting that the two groups were more or less equal in this area at the Give Aways level.

Interface specification is shown to be slightly better for the non-RKE group, but it is difficult to assess since the difference is only 0.3 between the two groups.

The ADD Process area denotes that the teams sought feedback about the ADD process itself at an abstract, conceptual level. Here we see an almost 0.7 difference between the groups in favour of the non-RKE teams. Upon examining the data more closely however, it does not support any conclusive interpretations.

4.2.2 Analysis of Architecture Quality

We now analyse the architectures to assess their quality differences between the RKE and non-RKE groups and how these relate to the findings from RQ1. It should be noted that this assessment is carried out prior to the system's implementation (i.e., coding, testing and installation). Thus, all judgements on the quality of the architectures are based on the information available as of the time the architectures were deemed to be completed.[15]

Table 2-6 shows the various critical project topics pertaining to architecture development and the averages for the two groups out of a possible score of 100. The overall quality score was 78.4 for the RKE groups, and 62.4 for the non-RKE groups. This result was analyzed through statistical testing and was found to be statistically significant (p = .035, see Section 4.1 for more detail). We now analyse below the results from the detailed assessment of each topic[16].

The category *module decomposition* includes: functional separation of concerns; *pattern determination* based on the quality drivers for the system; and consistency of relationships among the modules. Examining the details of the architectures, we found that one sub-area that the RKE group excelled at was the

[15] It is quite plausible that architectures are assessed subsequent to the implementation of a system. This, however, is another perspective of architecture assessment and it involves many other factors not necessary in our study, such as design, coding and testing and how these might have affected the architectural design.

[16] The detailed topics in Table 6 were not statistically tested because the study was controlled only at the overall level, and not at the detailed topic level. For example, if a non-RKE group did not perform well at *behaviour models,* this could be because they lack a type of knowledge, or that they did not have time to do this task because they struggled through the initial part of the project. Only through further, more tightly controlled experiments can causal inferences be made at the detailed topic level.

patterns work (on average, on a 7-point scale RKE: 4.5, Non-RKE: 3.5). They were better able to come up with a pattern that encompassed architectural *tactics* (i.e., design decisions) that did not introduce significant tradeoffs between the quality attributes, and therefore met the quality requirements for the system. This qualitative finding corroborates with the quantitative findings from the feedback data (see Figure 2-1, Figure 2-2, and Figure 2-3) where there was a difference in the *quality satisfaction, tactics,* and *patterns* work.

The *deployment* aspect of the architecture clearly favours the RKE group; they consistently performed better in all the criteria underlying this score (understandibility, readability, appropriateness of patterns to address quality issues, etc.). The underlying reasons, however, are not clear because of the lack of completeness in this area on the part of the non-RKE group. It could be that the non-RKE-group's processes did not enable them to complete this part.

The *component and connector* (C&C) views, unsurprisingly, were evaluated about the same for both the groups (62.7 for RKE, and 60.8 for non-RKE). First, we note that this topic is very architecture-centric (i.e., no background differential across the groups). Also, the 60's averages seem to reflect the fact that unlike module decomposition (which has similar properties to low-level design modelling) C&C is a relatively new topic for all the participants.

Overall Architectural Properties were assessed roughly the same (67.7 for RKE, and 64.6 for non-RKE). This category deals with how well various system views map to each other, dependence on COTS products, and the buildability quality attribute. Looking closer at these attributes, both types of groups were almost equal for mapping system views and buildability, which is to be expected since both are more solution-oriented attributes. The dependence on COTS products was not prevalent in this particular project and therefore both types of groups received a similar assessment.

Interface specification shows the biggest gap out of any of the areas in the architecture quality between the two groups (86.7 for RKE, 60.4 for non-RKE). This is quite surprising since there was no indication that this would likely occur from looking only at the feedback data (see Figure 2-1, Figure 2-2 and Figure 2-3), where both groups performed relatively equally. The principal reason for the gap could be that, the *interface specification* activity at first seems architecture and design oriented, without a clear or obvious link to the requirements. Yet, a "good" *interface specification* requires that (Bass et al., 2003) the natural language and modelled *functional requirements* be "parsed" into: (i) services that the modules (or components) must provide; (ii) resources these modules require; (iii) exceptions that can occur; (iv) quality attribute characteristics of the interface; and (v) the resources' syntax and semantics. It thus seems that those architects who are more "rooted" in the requirements, and also who can understand better the quality and functional implications of the requirements can create better *interface specifications.*

Architecture Topics	RKE Ave. (Out of 100)	Non-RKE Ave. (Out of 100)
Module Decomposition	74.5	66.
Deployment	78.9	47.2
Component and Connector	62.7	60.8
Overall Architecture Properties	67.7	64.6
Interface Specification	86.7	60.4
Behaviour Models	74.9	46
Descriptions	71.8	70
Architecture Reasoning	58.3	50.2
Documentation	73.7	59.7
Overall Architectural Quality	**78.4**	**62.4**

Table 2-6. Architecture quality comparison

Modelling the system's *behaviour* is another task that resulted in a wide gap between the two groups (74.9 for RKE, 46 for non-RKE). Much of the discussion from the *interface specification* can apply to this task as well. However, it is not quite as clear because three of the non-RKE groups did not submit any behaviour models, suggesting that they either did not know how to construct them at all (which is not very plausible, since this type of work is carried out similarly at low-level design which they have learnt in prerequisite classes), or that they simply did not have the time to complete because much work was invested in the other activities of the project. Since the feedback data is also inconclusive for this activity, more data is required to fully assess the difference between the two groups.

The score on *architectural descriptions* are slightly higher for the non-RKE group than for the RKE group (71.8 for RKE, 70 for non-RKE), though upon examining the architectures, it suggests that there is no trend for this slight difference between the two groups.

Architecture Reasoning score is higher for the RKE group, but overall is disappointingly low for both groups (58.3 for RKE, 50.2 for non-RKE). Some reasons are that both groups performed poorly in discussing *alternate decisions* to the principal design used, discussing the *quality attribute tradeoffs* that are introduced from their decisions, and documenting any *underlying assumptions* of the system. The criteria in which the RKE group performed better at was the documentation of how the chosen design satisfied quality attributes and how it implemented the tactics selected. This result again supports the trend of the RKE group understanding *quality* issues, *tactics* and *patterns* better than the non-RKE group.

Finally, the RKE group did much better than the non-RKE group (73.7 for RKE, 59.7 for non-RKE) in the architectural documentation, but it is difficult to claim this as an impact of the RE knowledge without further investigation.

4.3 Summary of the Findings

The preceding subsections discussed the results of the study's two research questions (RQ1 and RQ2); the first dealing with the quality of the final architecture submitted, and the second probing into the details of the first research question by investigating, in-depth, the product and process of the two respective study groups. The following are the key summary points:

- The RKE groups developed a better final architecture than the groups without RKE – the average on their assessment (out of a possible 100 points) was approximately 10 points higher for the RKE groups. This difference was found to be statistically significant.

- The feedback sessions (section 4.2.1) show that the non-RKE group sought more feedback than the RKE group in quality related categories such as *tactics, quality scenarios, the satisfaction of quality,* and *pattern determination.*
- Other areas of interest that emerged, although not as prominent, were the issues of *abstraction*, and of *allocating functionality to elements.* Both of these are not as "directly" linked to RE knowledge as the quality-related issues are, but they do have "indirect" ties to RE.
- The quality data also provided new areas of interest. *Interface Specification* was the task with the biggest difference between the two groups in terms of quality (86.7 for RKE, 60.4 for non-RKE). *Behaviour* and *deployment* modelling were also done much better by the RKE group.

We now proceed to discuss the implications of the findings.

5 Implications

The implications of the findings centre upon the areas of: hiring and training in the software industry, aligning RE and SA courses in the Software Engineering curricula, and methods and tools.

5.1 Hiring and Training

The response to research question RQ1 (see section 3.1) indicates that RKE architects outperform non-RKE architects. Thus, architectural training costs and architectural defects, at least in the early stages of an architect's career, can possibly be reduced by employing architects with the proper background in requirements. The detailed findings (see section 4.2) suggest that training for non-RKE architects could focus on the areas of *tactics, interface specification and pattern determination.* In addition to these areas, in (Ferrari and Madhavji, 2006) Ferrari and Madhavji identify specific requirements-oriented problematic areas for training architects (quality satisfaction, quality drivers determination, modelling quality requirements, abstraction, and requirements understanding).

Our long-term exposure with the software industry (in Canada) in the domains of database and information systems, systems software, insurance, telephony, games, utilities and the like, and phone interviews with practitioners in these domains, suggest that architects' roles are rarely filled consciously by agents with RKE. In many situations, they tend to have a technical background (databases, backup and recovery, platforms, etc.) which, while helpful for deploying architectures, still seems to leave an important gap in the front-end and more conceptual areas of architecting such as system structuring, determining dynamic models of the architecture, determining architectural patterns, dealing with potential quality attribute tradeoffs, among others.

Some practitioners indicated that requirements engineering, as pursued by pedagogy and research is still far from the reality of development practices in industry. For example, pedagogy and research tend to focus more on modelling requirements in specific notations and getting them consistent; whereas, industry tends to focus more on eliciting the right requirements (typically in a natural language), prioritising, and on the issues of costing, resources and deliverability.

Thus, RKE background amongst the developers in industry is not a common phenomenon and so this could be one reason why architect employees tend to have more technical (or implementation-oriented) background. However, with the field of RE increasingly penetrating higher institutions of learning through SE curriculum, there is hope that in the years to come the RKE "gap" amongst many architects in industry today may reduce, hopefully leading to higher quality of software systems.

5.2 Aligning RE and SA courses

As described in the introduction section, currently, there is considerable variability in the pre-requisites to the SA courses in post-secondary institutions. This can lead to: (i) implicit or unintended unfairness in courses where no allowance is made for students with/without RE background and (ii) difficulty in satisfactorily teaching both types of students at the same time. Certainly our own experience strongly supports this position.

Also, the IEEE/ACM curriculum for SE (Software Engineering, 2004) recommends only general SE or software construction as prerequisites for SA courses, depending on the core package selected. The general SE or software construction courses do not cover, in depth, the critical aspects, as highlighted in this study, of RE knowledge (e.g., *quality drivers, quality satisfaction and modelling quality scenarios)* that we have found to have significant impact on SA and so the current recommendations are less than ideal pre-requisites for an SA course. Whether a course on RE, in its entirety, should be a prerequisite (or even a co-requisite) to an SA course merits further investigation. Also, the "twin-peaks" model of life-cycle processes (Nuseibeh, 2001), where RE and SA are iteratively closely intertwined, is yet another consideration for organising RE and SA courses.

5.3 Methods and Tools

Lately, there has been some research interest in bridging the gap between RE and SA (STRAW, 2001 and 2003). Our findings in terms of targeted SA areas where RE has particular impact could thus help expedite this research both in the area of methods and tools. In (Bachmann et al., 2003b), they discuss the preliminary design for ArchE, a tool built to support ADD method. Currently, the tool supports decision-support for moving from quality scenarios to tactics for two quality attributes (*modifiability* and *performance*). However, based on the findings from this study, this tool (or other tool efforts such as GRL (Liu and Yu, 2003) and CBSP (Egyed et al., 2003), to name a couple), could possibly take advantage of specific linkages of the aforementioned *quality scenarios* and *tactics*, but also *quality drivers* and *architectural patterns* (see section 4.2) by enhancing decision support for non-RKE architects. For example, these tools could possibly capture the experience profile of their users to then automatically adjust to the varying architects potential needs. Likewise, the twin-peaks model (Nuseibeh, 2001) of life-cycle processes could possibly be refined further to give more detailed explanations of the inter-relationships between RE and SA. For example, explicit consideration can be made for RE and SA tasks where there is a particularly strong dependence between the two areas (such as *quality drivers determination*, *tactics usage*, and *modelling quality scenarios* to name a few). Besides, improvement of existing architecting methods (such as the ADD process (Bass et al., 2003), Preskriptor Process (Brandozzi and Perry, 2003), and CBSP (Egyed et al., 2001) could consider incorporating sub-areas where both RKE and non-RKE architects experienced significant difficulties (such as *patterns* and *component and connector views).*

6 Future Work

While the findings from the study described in this paper are interesting, they should be considered only a humble beginning, for there are many new areas for future studies. Here, we highlight a sample of these:

- A replication of this study would be critical to either refute or support the trends that emerged in this study's findings. Indeed, many such studies need to be conducted until firm conclusions can be drawn. Furthermore, replication within industry (though highly unlikely at least due to resource and time constraints) and architecting different application domains would be invaluable for generalisation of the findings to wider contexts and different types of systems.
- In our study, we did not track the feedback on requirements changes made after releasing them to the architects and, likewise, the impact of these changes on the architecture. Empirical work is lacking in this area and it would be potentially beneficial to see how the evolving system requirements would affect the architecting process.
- While this study primarily looked at RE knowledge and its impact on architectural *technical* activities, there is still a strong behavioural aspect to architecture development and requirements engineering such as communication among the various stakeholders, understanding customer needs and market trends, assembling and managing development teams, among others (Bredemeyer and Malan, 2006). Many of these skills are more human related, and less technically-oriented than what is needed in other software development phases (such as coding and testing). A strictly behavioural study would be useful in empirically providing skill and personality-aptitude sets for determining the "right" people for carrying out RE and SA, and also providing improvements in the human communicative

aspects of these areas (Curtis et al., 2001). This would provide empirical support for the discussions raised on this topic in (Clements et al., 2007).

7 Conclusions

The fields of Requirements Engineering and Systems Architectures are recognised to be amongst the most critical areas of software development, and recently there has been much interest in transitioning from requirements to architectures (STRAW, 2001 and 2003). In this paper, our objective was to investigate how, when architecting systems, the architects with software requirements knowledge and experience compare against those without. In particular, we conducted an empirical study involving 15 teams, collecting and analyzing data from diverse sources such as documented architectures, decision templates, emails, logs, and feedback sessions.

From the findings of the study, we conclude that architects with requirements knowledge and experience (RKE) perform better in terms of architectural quality, than those without RKE (in our study, it was by 16% (RKE: 78.4% and non-RKE: 62.4%), see Table 2-4). This difference was found to be statistically significant at a 96.5% confidence interval.

There were two data sources used to provide details into the relative performance of the specific technical areas: feedback sessions and final architectures. Based on our analysis of feedback (see section 4.2.1), the specific technical areas where RKE group excelled were: *tactics, quality scenarios, the satisfaction of quality,* and *pattern determination.* Looking more closely at the details in the final architectures produced (see section 4.2.2), two new areas emerged where the RKE group excelled: *interface specification* and *behaviour modelling.*

These findings can have important implications for hiring and training in the software industry, pedagogy, and architecting methods and tools, as described in section 5. For example, for hiring software architects, background analysis can be used as a discriminator between those with requirements knowledge and experience and those without. Likewise, for training in the area of systems architectures, specific requirements-oriented material (see section 5.1) can be used to augment the training of those without requirements knowledge and experience. In pedagogy, systems architecture and requirements courses can be aligned appropriately (see section 5.2) to take advantage of the requirements-oriented knowledge for optimal student performance in systems architecture courses. Finally, methods and tools research in the area or improved transitioning from requirements to architectures can possibly consider the findings (see section 5.3) as requirements for designing and implementing these methods and tools.

Examples of further ideas for empirical studies that could extend this work are discussed in section 6. These are replication of this study; examining how requirements that evolve during architecting affect the RE and SA process; and a more behaviour-oriented study that could empirically examine optimal skill-sets for architects.

Since this was only one exploratory-based study in a particular context, it would be a mistake to generalise these results verbatim to other contexts (Zave, 1997). However, this does not diminish the importance of the findings described in this paper. Rather, more such studies are needed in this area to add to the currently meagre body of empirical knowledge on RE and SA.

References

Bachmann, F., Bass, L., Klein, M., 2003a. Moving from quality attribute requirements to architectural decisions. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 122-129.

Bachmann, F., Bass, L., Klein, M., 2003b. Preliminary Design of ArchE: A Software Architecture Design Assistant. Technical Report, Software Engineering Institute, Carnegie Melon University, CMU/SEI-2003-TR-021 ESC-TR-2003-021.

Bass, L., Clements, P., Kazman, R., 2003. *Software Architecture in Practice, 2nd edition*, Addison-Wesley.

Berander, P., 2004. Using Students as Subjects in Requirements Prioritization. Proceedings 7th International Conference on Empirical Assessment & Evaluation in Software Engineering, Keele University, Staffordshire, UK, pp. 95-102.

Berg, B. L., 2007. Qualitative Research Methods for the Social Sciences. Boston, Pearson Allen & Bacon.

Boehm, B., 2002. Get ready for agile methods, with care. Computer Volume 35, Issue 1, pp. 64 – 69.

IEEE SWEBOK, 2004. Guide to the Software Engineering Body of Knowledge: 2004 Version. IEEE and IEEE Computer Society project, available at http://www.swebok.org/.

Brandozzi, M., Perry, D. E., 2003. From Goal-Oriented Requirements to Architectural Prescriptions: The Preskriptor Process. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 107-113.

Bredemeyer, D., Malan, R., 2006. The Role of the Architect. Archticture Resources for Enterprise Advantage, Bredemeyer Consulting.

Campbell, D. T., Stanley, J. C., 1963. Experimental and quasi-experimental designs for research. In N. L. Gage (Ed.), *Handbook of research on teaching* (pp. 1-76). Chicago: Rand-McNally.

Carmines, E. G., Zeller, R.A., 1991. *Reliability and validity assessment.* Newbury Park: Sage Publications.

Carver, J., Jaccheri, L., Morasca, S., 2003. Issues in Using Students in Empirical Studies in Software Engineering Education. Proceedings of the ninth International Symposium on Software Metrics (METRICS'03), Sydney, Australia, pp. 239-249.

Clements, P., Kazman, R., Klein, M., 2007. Working Session: Software Architecture Competence. Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA '07), Mumbai, India.

Curtis, B., Hefley, W. E., Miller, S. A., 2001. People Capability Maturity Model (P-CMM): Version 2.0. Carnegie Mellon Software Engineering Institute technical report, CMU/SEI-2001-MM-001.

Damian, D., Chisan, J., 2006. An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. Transactions on Software Engineering, 32(7), pp. 433-453.

Creswell, J. W., 2003. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* Thousand Oaks, CA: Sage Publications.

De Vaus, D. A., 2002. *Analyzing social science data.* SAGE Publishing Ltd, London.

Easterbrook, S.M., Yu, E., Aranda, J., Fan, Y., Horkoff, J., Leica, M., Qadir, R. A., 2005. Do Viewpoints Lead to Better Conceptual Models? An Exploratory Case

Study. 13th IEEE International Requirements Engineering Conference (RE'05), Paris, France, pp. 199-208.

Egyed, A., Grunbacher, P., Medvidovic, N., 2001. Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach. First International Workshop from Software Requirements to Architectures (STRAW '01), Toronto, Canada.

Ferrari, R., Madhavji, N., 2006. Requirements-Oriented Problems While Architecting: An Empirical Study. 12th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '06), Luxembourg, pp. 81-96.

Ferrari, R., and Madhavji, N. H., 2008. Software architecting without requirements engineering knowledge and experience: What are the repercussions?. Journal of Systems and Software, Volume 81, Issue 9, September 2008.

Hall, B., 2004. Public Interest Anthropology (PIA)", University of Pennsylvania, http://www.sas.upenn.edu/anthro/CPIA/methods.html.

Hoest, M., Regnell, B., Wohlin, C., 2000. Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment, Empirical Software Engineering, pp. 201-214.

In, H.; Kazman R., Olson, D., 2001. From Requirements Negotiation to Software Architectural Decisions. Second International Workshop from Software Requirements to Architectures (STRAW '01), Toronto, Canada.

Kazman, R., Klein, M., Clements, P., 2000. ATAM: Method for Architecture Evaluation. Technical Report, Software Engineering Institute, Carnegie Melon University, CMU/SEI-2000-TR-004 ESC-TR-2000-004.

Kotonya, G., Sommerville, I., 1998. *Requirements Engineering - Processes and Techniques*. Wiley.

Liu, WenQian, Easterbrook, S., 2003. Eliciting Architectural Decisions from Requirements using a Rule-based Framework. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 94-99.

Liu, D., Mei, H., 2003. Mapping requirements to software architecture by feature-orientation. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 69-76.

Liu, L. and Yu, Eric, 2003. From Requirements to Architectural Design – Using Goals and Scenarios. Second International Workshop from Software Requirements to Architectures (STRAW '01), Toronto, Canada.

Mason, J., 1996. *Qualitative Researching*. SAGE Publishing Ltd, London.

Miller, J., Madhavji, N., 2007. The Architecture-Requirements Interaction. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 07), Mumbai, India, pp. 20-23.

Nord, R. L., Soni, D., 2003. Experience with Global Analysis: A Practical Method for Analyzing Factors that Influence Software Architectures. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 34-40.

Nuseibeh, B., 2001. Weaving the Software Development Process Between Requirements and Architectures. Second International Workshop from Software Requirements to Architectures (STRAW '01), Toronto, Canada.

Poort, E.R., De With, P.H.N., 2004. Resolving requirements conflicts through non-functional decomposition. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 04), Oslo, Norway, pp. 145-154.

Rapanotti, L., Hall, G., Jackson, M., Nuseibeh, B., 2004. Architecture-driven Problem Decomposition. Proceedings of the 12th IEEE International Requirements Engineering Conference (RE 2004), Kyoto, Japan, pp. 80-89.

Runeson, P., 2003. Using Students as Experiment Subjects – An Analysis on Graduate and Freshman Student Data. EASE'03 – Proceedings 7th International Conference on Empirical Assessment & Evaluation in Software Engineering, Keel, U.K.

Schwanke, R., 2005. GEAR: A Good Enough Architectural Requirements Process. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, USA, pp.57-66.

Shaw, M., 2003. Writing good software engineering research papers: minitutorial. Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), Portland, USA, Tutorial Session, pp. 726-736.

Sheskin, D. J., 2004. Handbook of Parametric and Non-paramteric Statistical Procedures. Chapman and Hall/CRC.

Software Engineering, 2004. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. A Volume of the Computing Curricula Series, August 23, 2004, The Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery.

Software Requirements to Architectures Workshop (STRAW), 2001 and 2003.

Sommerville, I., 2006. *Software Engineering.* Addison Wesley, 8th edition.

Thelin, Thomas, 2004. Team-based fault content estimation in the software inspection process. 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, Scotland, pp. 263-272.

Tichy, W.F. , Lukowicz, Prechelt, L., Ernst A., 1995. Experimental Evaluation in Computer Science: A Quantiative Study. Journal of Systems and Software, January, pp. 1-18.

Wang, Z., Sherdil, K., Madhavji, N.H., 2005. ACCA: An Architecture-Centric Concern Analysis Method. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, USA, pp.99-108.

Wieringa, R. J. , Heerkens, J., 2006. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. Requirements Engineering Journal, Vol. 11, pp. 295-307.

Wildt, A. R., Ahtola, O. T., 1978. *Analysis of covariance.* Quantitative Applications in the Social Sciences series #12. Thousand Oaks, CA: Sage Publications.

Wohlin, C., Hoest, M, Wesslen, A., 2000. *Experimentation in Software Engineering: An Introduction.* Kluwer Academic Publishers, Norwell, MA.

Zave, P., 1997. Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys, Vol. 29, No. 4, pp.315-321.

Zelkowitz, M., Wallace, D., 1997. Experimental validation in software engineering. Information Software Technology, volume 39, pp. 735-743.

Chapter 3

Architecting-problems rooted in requirements[17]

1 Introduction

Requirements! This expletive is obviously not meant to deny the importance of other kinds of software artefacts in a software project, such as architecture, design and code. Rather, it is meant to emphasise the ubiquity, not to mention the importance, of requirements when carrying out *non*-requirements engineering tasks. It is important to stress this point because a closer examination of the community's research focus on "requirements" suggests that, predominantly, the effort is being spent within the confines of the *requirements* engineering (RE) process (e.g., on ways to elicit, analyse and model requirements) and little on *other* software engineering processes where requirements are actually being *used*.

There are exceptions to this norm, however; in particular, the work on requirements traceability (Ramesh and Jarke, 2001); on bridging the gap between software requirements and architectures (STRAW, 2001 and 2003; Nuseibeh, 2001; Madhavji and Perry, 2004; Rapanotti et al., 2004) and on developing specifications that are well-suited for design and implementation activities (Zave, 1997). Besides such individual efforts, there is recognition also at the community level of the importance of requirements in relation to other processes. In particular, at the 14th Requirements Engineering Conference, 2006, there was a keynote address on the relation between testing and requirements (Graham, 2006).

In the quest to explore requirements issues in a non-requirements setting, we ask a rather rarely posed question:

[17] A version of this chapter was published in (Ferrari and Madhavji, 2008).

"*What kinds of requirements-oriented problems are being experienced while architecting a software system?*"

That is, the focus of this paper is that, during the software architecting (SA) process, if there are problems experienced by the architects then we would like to determine those subset of problems that are associated with (or rooted in) software requirements. In other words, we are primarily interested in investigating the *architecting* process and how requirements are treated in the context of this process. The focus of the paper is not in scrutinising the *requirements* engineering process directly or the artefacts being developed in that process.

While the essence of the posed question is also important for processes other than architecting (Bass et al., 2003), our focus on the architecting process is based on the fact that architecting is not only at the front-end of the development and evolution processes but is also tightly intertwined with the RE process (Nuseibeh, 2001) and so diminishing requirements-oriented (RO) problems there can have major (positive) quality, cost and time impact on the rest of the development process.

The practical value of the posed question lies in the kind of *feedback* that can be obtained from the findings that could precipitate improvements in the RE technologies, which is clearly of central importance to the RE community (Nuseibeh, 2001; Madhavji and Perry, 2004; Finkelstein, 2000). For instance, if the architects persistently have difficulty in understanding certain types of requirements, this feedback could drive ways to improve requirements *specification*, *documentation* and *communication* of these particular types of requirements tailored to the needs of the software architects. Likewise, if they have difficulty in ascertaining whether or not the emerging architecture will satisfy the desired qualities, this feedback could lead to improved linguistic mechanisms to describe the different quality drivers so that quality assessment is simplified. This way, the feedback obtained from answering the posed research question could play an empirical role in improving RE technologies which, in turn, could improve the handling of requirements outside the RE process.

Note that the RO problems experienced outside the RE process can stem from two key sources: (i) deficiencies (e.g., incompleteness, inconsistency, ambiguity, etc.) inherent in the requirements themselves and (ii) deficiencies in the RE technologies (e.g., notation complexity, expressive power, documentation meta-model, analysis tools, validation process, prioritisation techniques, handover process and others). In a development-focused setting, the first source would normally lead to *product*-oriented feedback, which could result in requirements fixes but not necessarily improvement in RE technologies; whereas, the second source would not normally lead to such feedback but, instead, would likely stifle development capability. In a research setting, however, both sorts of deficiencies could lead to *technology*-oriented feedback which, in turn, could lead to RE technology improvements. It is such feedback from both sorts of deficiencies, that is of particular interest in this paper because it is the resultant improvements from such feedback that could have a lasting impact in the RE field. Thus, a prerequisite to improving RE technologies is to understand the kinds of problems stakeholders face in using requirements to accomplish their goals.

The posed research question is, in fact, part of an empirical study involving sixteen teams, each architecting the same banking application from the same set of requirements. The architecting method used was Attribute Driven Design (ADD) (Bass et al., 2003). The study found that, for example, there were several different types of RO problems, of varying severity, which the architects faced in using the given requirements; that those architects with RE background also faced RO problems; and about a third of all problems were RO problems. When we shared our results with requirements analysts, architects and process specialists from a large insurance company, they concurred with our findings with their experience within the company. There were also some areas where there were relatively few RO problems.

The paper also describes some implications of the findings for the RE field, particularly in areas of: expression of quality requirements for different stakeholders; empirical studies on quality scenarios; tighter integration of RE and software architecting (SA) processes; and requirements to architecture mapping. To our knowledge, the current literature does not describe any empirical studies on RO problems in other development areas, especially architecting. In this sense, this study is the first one of its kind and acts as a precursor to potential other studies that can focus on replication.

In the next section, this paper describes the case study, section 3 describes the findings and discusses the implications of the results for the field of RE. Following this, section 4 describes related work, section 5 describes possible future studies involving hypotheses that have emerged from this study, and section 6 concludes the paper.

2 *The Empirical Study*

In this section, we first describe the study design. Following this, we discuss the participants of the study; system requirements; the architecting project; and the research procedures used.

2.1 Study Design

The type of study conducted was a *multiple-case* study design (Creswell, 2003). There were sixteen parallel cases (i.e., one case per architecting team) in our study. We also had a degree of control in our study in that we separated the participants into two groups: one that was composed of teams consisting only of requirements engineering knowledgeable participants (the RE teams), and the other group composed of teams without this knowledge (the non-RE teams). This was determined through a background questionnaire, where a series of "Yes/No" questions relating to their academic and industrial background were given. The participants who answered "Yes" for academic background were later checked to confirm that they had all taken a Requirements Engineering course, where they had learnt such topics as requirements elicitation, modelling, analysis, prioritisation, validation, among others. No participants had reported any industrial experience. Also, through the numerous interactions with the subjects during the course of this study, at no time was there any doubt concerning any specific subject as to whether he or she was mis-classified. There was thus a clear split between those who had requirements knowledge and those without such knowledge.

Despite the control we had in the study, it was an *exploratory* study in that we had no initial hypothesis, and we did not know which phenomena were important (i.e., the types of RO problems experienced by the architects). This is because, to our knowledge, there wasn't much background literature related to the posed research question. Though the exploratory nature of the case study is well suited for analysing the commonality and differences across cases that have similar traits (Creswell, 2003), it is important to note that the primary focus of the study was on discovering the major RO problems while architecting a system, not on making comparisons between the two groups.

2.2 Participants

We used availability (or *convenience*) sampling (Creswell, 2003), where the participants were drawn from the final year *Software Architecture* course at the University of Western Ontario (UWO). There were sixteen architecting teams, each comprised of four members.

2.3 The Requirements Document

The system to be architected was in the "banking" domain. The application included three different modes of banking for the clients: ATM, internet banking and telephone banking services; access and reporting features for the banking staff; client and financial database; various quality drivers, such as security, availability, performance, usability, maintainability, and others. In all, there were some eighty-five high-level requirements to contend with, which is sizeable. The requirements followed the organisational structure as found in (Somerville and Sawyer, 2000). This included a document preface, which described the organisation and the business needs, followed by the actual requirements with rationale written in natural language. The requirements section was split into different sub-sections each detailing requirements for a given subsystem, along with requirements that described properties that the overall system should have. Prior to the start of the architecting project, the architects were given a session where the project was described, including the application domain and the format and structure of the requirements, and any questions or concerns were addressed.

The requirements for the system were obtained from an external source. Prior to conducting the study, these requirements were validated by a team of five people for acceptability in general, "and for any serious or obvious flaws"; the semantic-content of the document was not altered. The result of this process is that a few grammatical fixes were made, along with the elaboration and clarification of certain requirements.

The validators had requirements, architecture, and software engineering experience ranging from 3 to 27 years. We did this in order to reduce *researcher bias* in the study. We also did not want to "fix" the document to the point where it was considered “perfect”. In a real-world setting, the requirements documents given for architecting or system development are not always “perfect”, and we wanted to emulate this by delivering an acceptable document to the participants.

2.4 Architecting Project

Given these requirements, as mentioned in the introduction section, each of the sixteen teams developed an architecture from the same requirements using the ADD method[18] (Bass et al., 2003). The projects were all conducted at UWO. The key steps of the ADD method include: understanding the requirements and developing the quality scenarios if they do not already exist; iteratively decomposing a selected module, choosing architectural drivers from the scenarios and functional requirements, choosing or creating an architectural pattern (using appropriate tactics) that satisfies the architectural drivers, identifying child modules to implement the tactics, instantiating the modules with functionality, defining interfaces, verifying and refining use cases and quality scenarios and making them constraints for the child modules.

Each team had to develop and document the system architecture and, in the process, capture, in the defined templates, such items as: design decisions, rationale, underlying assumptions, issues arising, resolution of items, etc. In addition, each team had the freedom to seek help on any difficulties they faced during their project. We termed these “feedback” sessions.

There were always two researchers conducting these feedback sessions. One researcher was present in all the sessions to ensure that the sessions were carried out in a consistent manner. The second researcher contributed to the feedback interactions as and when necessary. The researchers involved had no direct investment in the study; this was done to reduce *researcher bias*. Also, there were two levels of management inherent in these feedback sessions. One was of a global nature to ensure that feedback sessions were scheduled and held across the teams, that the process was running smoothly, and that, finally, data was transcribed and met

[18] We used the ADD method for this study because the context (Architecture course at UWO) in which the study was to be conducted already had the ADD method established in terms of both

quality requirements. The second level of management was more local to a particular session to ensure that team members were present in the appropriate session, data was properly recorded in the equipment, and that it was logged in a database. All the feedback sessions were recorded and later transcribed. More details on this process are given in Section 2.5.1.

A possible threat with the architecting project is that the participants' architectures produced were *conceptual*. That is, there was not to be any implementation and so certain static properties (such as fitness between the architecture's structure and allocation of code components) and dynamic properties (such as delivery of performance, security, availability, etc.) couldn't be checked through actual implementation and operation of the system. This implied that there were no end-user consequences of the architectural decisions being made, though there were clearly academic-performance-related consequences. Thus, these differences should be born in mind when attempting to generalise results from this study to other domains. However, in order to mitigate the threat of the "quality of the results", weekly motivational meetings were held where the participants' decisions were reviewed by the researchers on an on-going basis, and feedback for improvements was given.

2.5 Research Procedures

In this section, we describe the research methodology that was used to conduct the study. First we discuss the data collection method, second the data analysis procedure, and then we describe validation checks that were conducted on our empirical procedures.

2.5.1 Data Collection

In theory, there are two "areas" where data and information pertaining to requirement-oriented (RO) problems could be collected: *product* (e.g., the documented architectures) and *process* (e.g., intra-team email communications, data templates and feedback sessions). In this study, we focused expressly on the process and there is a good reason for this.

material and teaching resources. Using another architecting method in the course was not an option.

Analysis of the final architecture (or product) quality was not used to judge RO problems because the architecting process is much "closer" to the RO issues (e.g., understanding and consideration of quality drivers and relating them to architectural choices); whereas, the documented architecture is much "farther" away from the RO issues in that the architecture is a culmination of many different elements such as various modelling techniques used; documentation organisation; and degree and clarity of the documentation on the tactics used, allocation of functionality, deployment, views of the architecture, design choices and rationale, etc. Thus, it is difficult to discern from the final documentation the kinds of RO problems the teams had encountered. Rather, in this study, we were concerned with the process work that was carried out and the RO problems that were encountered along the way. In practice, it was much simpler to gather relevant data while the process was being conducted.

The data-set gathered in the study was quite extensive – approximately 50 hours of recorded interactive feedback sessions (see section 2.4 above), which were subsequently transcribed by three domain experts and verified for accuracy. Beyond this, there were numerous email communications and many data templates which were also explored to identify in-process RO problems.

Ethnographic methods (Hancock, 2002) were used such as participant observation and semi-structured interviews. These methods focused on gathering rich and detailed data regarding any possible topic or theme that arose within the research domain to be investigated, thus complying with the *exploratory* nature of the study.

2.5.2 Analysis of Feedback

To assess the issues that arose in the feedback sessions, we carried out *content analysis* (Mason, 1996) on the transcribed data. In essence, the frequency of the various types of feedback (i.e., severity of RO problems, and technical activity in the architecting process) was counted. The technical activities were identified beforehand from the ADD architecting process (Bass et al., 2003) and relevant research literature, and appropriate categories were formed *a priori* and validated by six experts over several iterations. These categories[19] are shown in Table 3-1.

To determine the type of feedback, *thematic coding*[20] was done on the entire data set, using the *a priori* categories of the architecting process and discovering any new *inductive* categories while coding (Mason, 1996). QSR's NUD*IST 4.0[21] was used for *thematic coding*; it manages and stores all the emerging codes, and simplifies retrieval of text units that have been coded.

No new categories of the architecting process were discovered and that our initial list of *a priori* categories covered the breadth of problems that were experienced during the architects process. However, the analysis resulted in the identification of three levels of severity of RO problems experienced by various teams: "*Mild*", "*Moderate*" and "*Severe*". These categories were *inductive* in that they were discovered after the data collection phase.

The definitions of the severity levels, with examples, are:

"*Mild*" - These were interactions where the participants only had a mild problem; little feedback was required to proceed with their architectural design. An example about modelling quality requirements (see **Table 3-1**) from the data (P: participant; R: researcher) is:

P: *Do our concrete scenarios help us in shaping our architectural patterns?*

R: *Not directly so, concrete scenarios are there to identify things that will happen often in the system, and with those identified you can prioritise what is important in the system.*

"*Moderate*" – These are interactions where the participants asking a question about a given topic needed a detailed explanation in order to proceed further in their solution design. Another example of modelling quality requirements (see **Table 3-1**) is:

[19] There also were 15 non-RO categories (mainly architectural), which are not relevant to this paper but examples of which can be found in (Ferrari and Madhavji, 2007).

[20] *Thematic coding* is a qualitative data analysis procedure where the researcher develops categories of concepts and themes that emerge from the data source. It is an 'open' process in that the researcher makes no prior assumptions about what the findings may be.

[21] QSR NUD*IST 4.0. QSR International Pty Ltd., 1999. Available at

P: *We weren't sure how specific we should be getting with the quality scenarios. Here's a few that deal with availability, but some seem to be specific, like this one about a power failure, which will not happen often. We're just confused with these scenarios.*
R: *So this is something that would happen a lot so that's the purpose of the quality scenarios, you're just thinking of ... obviously there's hundreds of scenarios you can think of in any given system, but, you're trying to think of those that are most important in the system, the ones that will be encountered the most often.*

"*Severe*" – These are interactions where the participants asking a question about a given topic needed a full solution to a particular problem for them to move forward in their design. A modelling quality requirements (see **Table 3-1**) example of this is:

P: *we kind of just did the drivers as the requirements dictated, the ones that seemed the most important. Is that the way it's supposed to happen? I was a little worried about that, so do you want say performance, and then just say why you do based on the requirements?*
R: *ummm, yeah, you could use the requirements as an example of why you consider performance a key driver. I mean that's where quality scenarios are critical. Scenarios deal with quality issues, non-functional, and so really the scenarios, I mean you could come up with hundreds of them ... but you're trying to come up with the ones that seem the most important based on the requirements and your own banking knowledge. And then, you can use these scenarios and say ok, so this will happen a lot, so it is high priority. Once this is done, you come up with tactics to resolve those scenarios. But I mean a scenario is attached to a quality attribute. So really, it's just a way of prioritising, your key quality attributes.*

Category	Definition
Requirements Separation	Deals with separating the functional and non-functional requirements.

http://www.qsrinternational.com

Requirements Understanding	This task involves understanding specific, individual requirements, as well as the set as a whole.
Domain Understanding	Deals with the understanding of the application domain, in our case, the domain of electronic banking.
Use Case Modelling	Use case models illustrate the units of functionality provided by the system.
Constraints	This relates to working with requirements that act as constraints on the system but not necessarily architectural properties. Examples include requirements that deal with coding and low-level design standards, process requirements, and software testing requirements.
Context Modelling	The aim here is to model the system to be built as a 'black box', and to show how it interacts in the environment in which it is to exist.
Quality Drivers Determination	This is the activity of deciding the key architectural quality drivers from the set of given requirements.
Modelling Quality Requirements	This is complementary to modelling *functional* requirements, involving, amongst other things, a stimulus (e.g., a change request of a certain type), a response to this stimulus (e.g., changes made to certain components) and response measure (e.g., estimated time for that type of change) (Bass et al., 2003). The resultant quality scenarios help understand, specify and prioritise the desirable system qualities. They are a trigger for architectural design and they provide a means to check that the architecture satisfies the intended quality attributes.
Quality Satisfaction	This task involves discerning whether the architectural solution would, or did, meet the quality requirements.
Reasoning	This activity deals with the thinking, expressing and rationalising about architectural decisions made in terms of the functional and quality requirements.
Abstraction	Some requirements were documented at a higher-level of abstraction; whereas, some others were broken down to finer levels. These multi-level requirements often were related functionally, which meant that for architecting purposes the mapping between multi-level requirements and components needed to be controlled through component hierarchies and interface descriptions.

Table 3-1. Feedback categories.

This analysis procedure was validated, as discussed in the next section.

2.5.3 Validation of Coding Procedure

A single researcher executed the coding procedure initially. After two teams' worth of feedback was analysed, two researchers in the Social Sciences area (where qualitative coding is more commonly used) reviewed the work for accuracy of the coding procedure. The feedback from these researchers was used to further train and refine the lead coder for doing the work. The feedback included being aware of the multiple levels of severity, which eventually led to the three defined levels: mild, moderate and severe. With this feedback, the "test-retest" method (Metze, 2001) was used to ensure a high level of reliability. This is an inter-rater agreement method where two raters are used and wherever there is disagreement between them, that "part" of the data is retested by at least one other rater until an agreement is reached.

All the data was analysed and "transformed" into frequency counts. Following this step, a second researcher reviewed the analysis to check for flaws in the coding process. The reviewer would take a number of coded items in the text and state agreement or disagreement on the assigned code. If there were any major disagreements, then a second reviewer was brought in to reconcile the disagreement and come up with a code for the given text. In the end, all the data was coded with agreement. In all, over 60 hours were invested in the coding analysis, and another 15-20 hours to conduct the validation.

2.6 Threats to Validity

As described in section 2.1, our study is *exploratory* and therefore we are not specifically looking for *causal* relationships with respect to our study constructs (requirements oriented problems). Thus, we do not discuss threats to the "internal" validity of our study. Internal validity is the extent to which the findings of a study accurately represent a causal relationship between an independent variable(s) and the dependent variable (or outcome). We discuss in the following section typical qualitative study threats and also discuss the external validity and, in particular, the generalisability of our findings to other settings and contexts.

2.6.1 Qualitative Validity

In qualitative studies, a validation technique, called *triangulation* (Guion, 2002), is used to ensure validity in the study. *Triangulation* is a method of establishing the accuracy of a study's findings by comparing three or more types of independent points of view on a given aspect of the research process (methodology, data, etc.) (Guion, 2002). There are different types of *triangulation* that can be used together to form a strong basis of validity. In this section, we will discuss how we used three different types of *triangulation* to ensure validity in our study. The *triangulations* used were: *data triangulation, methodological triangulation,* and *investigator triangulation.*

2.6.1.1 Data Triangulation

Data triangulation is the use of different sources of data/information on which the study results are based. If there is consistency in the data/information provided across the various data sources that are used, then this suggests that the data is valid. In our study, as mentioned in section 2.5.1 (Data Collection), our data-set came from numerous sources including the feedback sessions, intra-team e-mail communications, and various data collection templates that the participants had to complete. Although the volume of data provided by each of these sources was different (feedback sessions provided the most data, then the templates followed by the e-mail communications), the proportion of the *types* of requirements-oriented problems were quite similar in each of the sources.

2.6.1.2 Methodological Triangulation

Methodological triangulation is the use of different methodological techniques (that could be either quantitative or qualitative) in the study and, if the conclusions from each method are consistent, then validity is increased. In our study, we used various qualitative methods such as participant observation, semi-structured interviews, document analysis, as well as quantitative content analysis. The resultant data from these various methods, and its subsequent analysis, showed similar conclusions, that architects experienced RO problems when architecting a system (see section 3). This consistency establishes methodological validity in our study.

2.6.1.3 Investigator Triangulation

Investigator triangulation is using several investigators/researchers in the conduct of the study and all its processes. In our study, at every stage in the process (e.g., data collection, data analysis, research question validation, etc.), we used multiple researchers to actually perform the processes as well as validate them. The findings observed from each researcher were compared to ensure that their conclusions were similar and therefore we conclude validity was reached.

2.6.1.4 Ecological Triangulation

Another type of triangulation that exists, but which we could not attain, is *Ecological Triangulation.* This is when the study is conducted at many different settings and places, and then the findings from each of these settings/places are compared to see if they are similar. This type of triangulation can be attained for this study through replication of this study in other contexts (e.g., in industry).

Without first replicating this study, it is difficult to immediately generalise the results to other contexts. However, this research provides a necessary groundwork for further studies of this kind. In the next section we further discuss validity, specifically the *external validity* threats to our study.

2.6.2 External Validity

External validity is the degree to which any findings from the study can be "generalised to and across populations of persons, settings, and time." (Creswell, 2003).

2.6.2.1 Population validity

Using students as participants in our study is a threat that is directly imposed on the *generalisability* of the findings to industrial contexts. This is a common risk in ethnography based studies but recent research in Software Engineering (Host et al., 2000; Runeson, 2003; Thelin, 2004) have shown that senior-level students perform similarly to "novice" software engineers with one-two years industry experience. Also, in (Berander, 2004), the use of students is promoted when conducting an investigation that has not been studied much before, such as in our case. Studies with students can provide early indications of trends, and preliminary evidence prior to committing to conducting studies in industry.

2.6.2.2 Ecological validity

This threat refers to the generalisability of the study results across all settings. As with *population validity,* the academic setting can be quite different from an industrial context so the threat is present. However, the project was loosely structured (as opposed to a strict "laboratory" setting) so that the setting could more closely mirror real-world work.

We omit the *temporal validity* threat here because there is little reason to believe that the results of this study could not be generalised over time given the current set of requirements methods, tools, processes, etc. that requirements engineers use.

To determine the alignment of our study's findings with that of industry, we conducted an external validation session that is described in the next section.

2.6.2.3 External Validation

Following the analysis of the results, we had an approximately three-hour interactive session with two senior practitioners from a large insurance company. The purpose of this session was to share our findings with them and to obtain their views on the findings in the context of *their* work environment (external validation). One practitioner was the head of software development processes and technologies and had a mandate to improve these processes and technologies in the company. She was also heading the Quality group. The second was a requirements expert, linking business needs to software development. Both the agents were with the same insurance company for over fifteen years.

The validation session proceeded by first presenting the company agents with our research context and briefing them generally on the research projects underway at UWO. We then presented the goals of the empirical study described in this paper, the study context and design, and the findings and implications. Following this, we discussed, in turn, each finding and its implication and asked the agents whether or not the finding had any validity in their work context and whether the implication had any relevance to them. We took notes of their views.

Separately, we also interviewed a senior developer/architect (with over 20 years of experience) from the same insurance company and basically underwent a similar validation procedure. Their collective feedback is described in this paper in the next section along with the results.

3 Results, Interpretations and Implications

In this section, we present the various findings, interpret the results and describe their implications for the field of RE. Each of the findings is modularised in that the results, interpretations and implications are discussed all together for each of the points.

We begin by first examining the overall picture. Requirements-oriented (RO) problems constituted 35% of the total problems encountered in conducting the ADD architecting process. This number indicates that despite working on the solution side of system design, the architects had significant RO difficulty in conducting the tasks – which suggests that it merits further analysis if not action.

Table 3-2 shows the frequency distribution of the varying levels of severity of problems across the sixteen teams.

Team #	Mild	Moderate	Severe
1	9	4	1
2	1	2	0
3	10	8	1
4	4	1	2
5	6	6	2
6	18	10	4
7	3	9	0
8	6	7	0
9	7	9	0
10	4	1	1
11	8	4	1
12	11	3	0
13	4	13	2
14	21	21	8
15	12	9	2
16	10	12	1
Mean	8.38	7.44	1.56
Std. Dev.	5.37	5.21	2.03
Max	21	21	8
Min	1	1	0

Table 3-2. Distribution of RO problems by severity across all teams.

We see that teams 14 and 6 encountered most problems; whereas, teams 2 and 10 encountered fewest problems. The mean and the standard deviation values suggest that, overall, the RO problems are not isolated incidences. This should therefore raise some concern in the RE community.

3.1 RO Problems in Technical Areas

The pie chart in Figure 3-1 shows the identified RO problem areas and their magnitudes. The most problematic areas were:

- *Quality Satisfaction* (22%)
- *Requirements understanding* (18%)
- *Quality drivers determination* (15%)
- *Abstraction* (14%)
- *Modelling quality requirements (scenarios)* (12%)

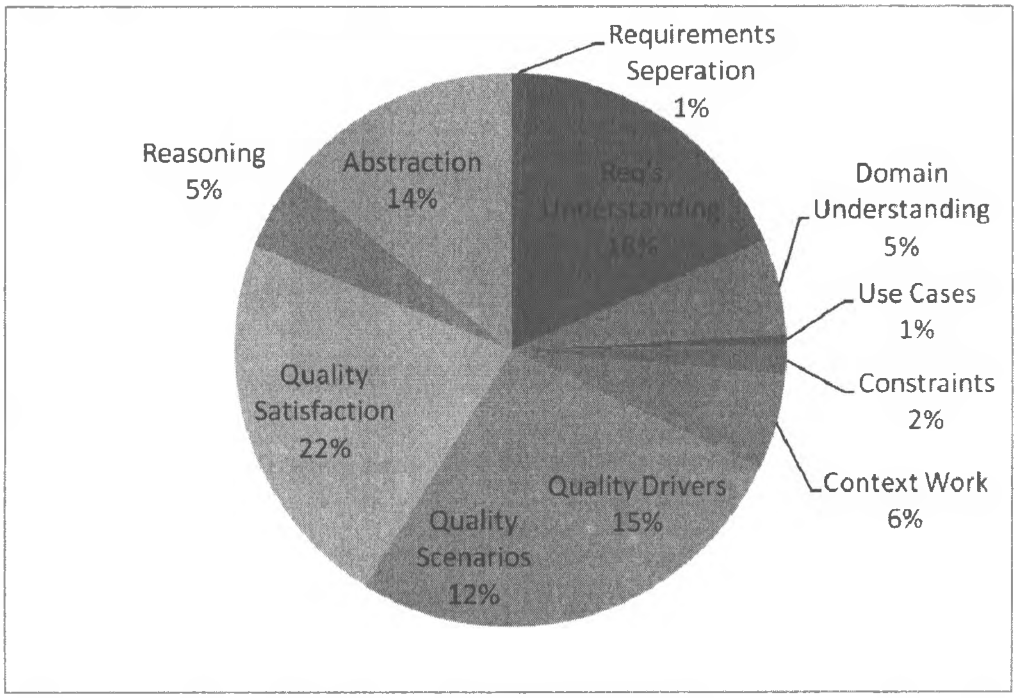
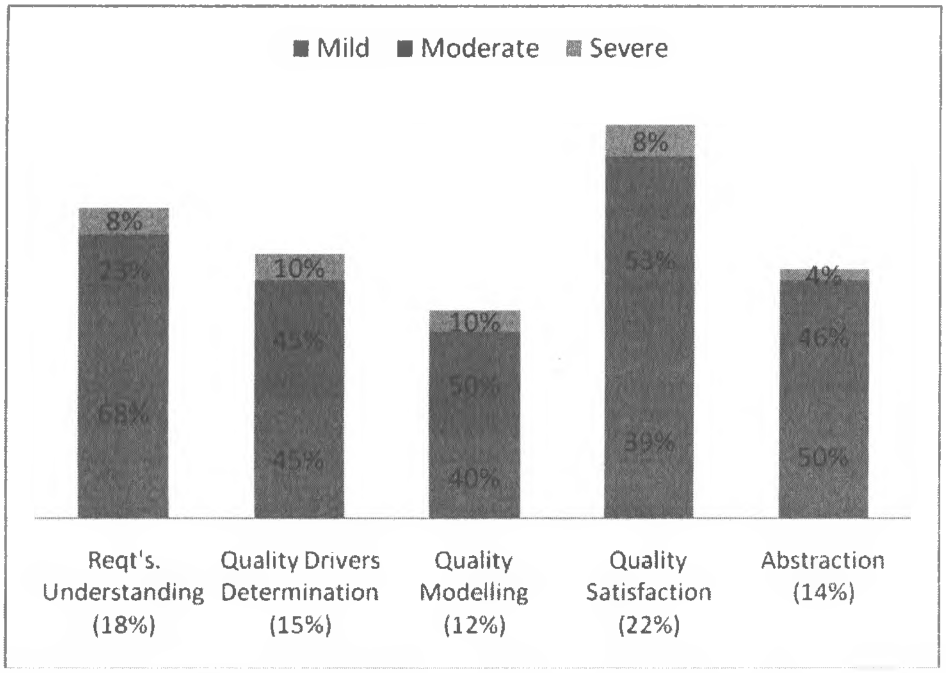


Figure 3-1. RO problems areas.

Figure 3-2 shows the severity levels of the problems among the top five problematic areas. In four of the five categories (except abstraction), 8-10% of the problems were severe. Also, other than the Requirements Understanding and Quality Satisfaction categories, there was an almost even split between moderate and mild number of problems in the remaining categories. This suggests that these areas were quite problematic for the architects; they did not only face mild problems. The

Requirements Understanding category was dominated by mild problems. The detailed severity data supports that Quality Satisfaction is the most problematic category with over 50% of its problems being moderate. This rendition of the problems seems to prioritise the five key problem areas into three main buckets: Quality Satisfaction; Quality Drivers Determination, Quality Modelling and Abstraction; and Requirements Understanding. Each of these problematic areas are discussed below.



The bars in the chart show the five key problematic areas from Figure 3-1. A given bar also shows the relative volume of problem spread across the three severity levels (rounded to zero decimal places).

Figure 3-2. Severity levels in the key RO problematic areas.

<u>Quality Satisfaction (22%)</u>: is the ability to discern whether the architectural solution would, or did, meet the quality requirements. This analysis was done where appropriate in the ADD process, for example whenever a pattern was formed/selected based on the key quality drivers and the tactics used to meet the quality demands. Often times when the quality was considered to be unsatisfactory, the architects refined and re-prioritised the requirements or the quality drivers with appropriate consultation with the stakeholders.

When examining the data closely, most of the problems seem to lie with *performance* and *availability* requirements; *security* and *modifiability* were the other quality attributes that were typically considered to be of the highest priority, but these were implemented and reasoned with relative ease. This finding was shared by industry experts in the insurance business when we discussed our results with them. However, their experience suggested that the performance requirements involving hardware equipment are simpler to deal with than those that are purely software or conceptual in nature. In our study, we were dealing with the latter type of requirements.

This suggests that there is a certain type of property in requirements that makes it simpler (or harder) to relate the requirements to an architecture. That the way the different types of quality requirements are expressed can lend themselves to the different degrees of understanding by the developers. For example, *security* requirements will often express, tangible, concrete functions involved in security matters, e.g., *access to services* in:

R*3.3 Customer should be provided access to internet banking services based on valid bank account number, user defined password, and access permissions set out for the bank customer.*

which can then be mapped to specific elements in an architecture with relative ease. Likewise, *modifiability* has long established principles of information hiding, cohesion and coupling which help in system structuring to localise change; issues that are closely tied to the structure of the system and are therefore architectural issues.

Conversely, performance requirements do not readily suggest specific, implementable elements except perhaps those involving specific physical elements. The following example from the requirements document is by no means flawed or defective, yet there were many difficulties encountered when using such a requirement:

R1.18 System must complete a transaction in less than three seconds. This assumes a direct connection by an employee. For other services, like Internet banking, this time could be different because of external factors like the user's connection.

Whereas, for example, in the software testing process the above performance requirement can be (more or less) related easily to specific test cases, the same requirement does not lend itself readily to any architectural decisions.

The difficulties that arose did not only lie in the initial phases of the architecture design, but also at the back-end of the architecting process when reviews were performed to check whether the quality requirements were satisfied. This indicates that the RO problems can penetrate deeply into non-RE processes and may not lend themselves easily to a "quick-fix" solution outside the RE processes.

Previous work described in (Nixon, 1993) touches upon this by proposing a means of specifying and implementing performance requirements. The Twin Peaks lifecycle model (Nuseibeh, 2001) also suggests that requirements and architectural design issues need to be brought closer together to simplify design-fitness assessment. Our industrial associates are attempting to deal with such closer integration but have not concluded as yet on which approach to pursue. Although our quality attribute coverage is not exhaustive, the results do suggest the need for further research in the way different quality attributes could be expressed for the different types of users.

Modelling quality requirements (12%): is complementary to modelling *functional* requirements, involving, amongst other things, a stimulus and a response to this stimulus (Bass et al., 2003). This activity is done at the start of the ADD process and the resultant scenarios act as input into the architecting process. These quality scenarios are refined (changed or removed) during the architecting process as problems arise with tradeoffs being introduced with key quality drivers. The resultant quality scenarios help understand, specify and prioritise the desirable system qualities. They are a trigger for architectural design and they provide a means to check that the architecture satisfies the intended quality attributes. The relatively high frequency of difficulties encountered in this area corroborates with the tightly related area of *Quality Satisfaction* described above. It seems that theory is ahead of practice in this area at the moment, which should be a motivation to conduct empirical studies. For example, our industry associates are struggling with modelling quality requirements in several ways: from getting the *idea* accepted to getting quality models institutionalised into their processes, and ensuring that there is satisfactory coverage of the quality scenarios. An implication of this could be to conduct empirical studies in industrial contexts on the modelling and use of quality scenarios to assess their practicality.

Quality Drivers Determination (15%): This is the activity of deciding the key architectural quality drivers from the set of given requirements. Because not all qualities can be realistically satisfied in a given design, they tend to introduce tradeoffs, which implies that prioritisation of the qualities is required. This step is carried out prior to entering the ADD process and it provides input to the ADD process. However, it is also carried out as architectural patterns are determined (while iterating through ADD), and tradeoffs introduced from the patterns suggest that quality drivers should be modified.

Individual requirements, however, already have one or more quality drivers associated with them as part of their specification. Thus, when it comes to architecting, several, related requirements need to be considered together to form a set of interacting components. This is when the conflicts and tradeoffs among the competing quality drivers arise. But note that requirements-level conflicts and tradeoffs, and prioritisation of quality drivers are usually done at RE time. An implication of this is that the results of the RE work, including the underlying assumptions and the supporting rationale should thus be made available to the architects in the hope that they would have a head start in their processes. This can be in the context of a "handover" process or something similar to SEI's Quality Attribute Workshop (QAW) (Barbacci et al., 2003). In the case study, there was a disconnect between the RE and SA processes in this respect. A tighter integration between these two processes is therefore quite appealing (Nuseibeh, 2001). Our industrial associates' experience is that while requirements rationale was generally passed along to the architects, it was not adequate; the architects still struggle with the lack of domain information (e.g., assumptions), alternative strategies, and documents about unresolved issues or to be aware of certain pitfalls.

Abstraction (14%): Yet another area of difficulty experienced was *Abstraction* in the requirements document. As described earlier, this was to do with varying levels of abstraction of the different requirements. This activity, of mapping requirements to architectural components, is done throughout the ADD process, but perhaps where it is most prevalent is the step of ADD that involves allocating functionality (specified in the requirements) to the software architecture. With subjective judgment on the levels of abstraction of the requirements, it seemed to leave room for mapping problems between requirements and component hierarchies in the architectures amongst some architects.

An implication of this problem might be the need to ask or verify, during requirements engineering or architecting, whether the requirements have been documented at a level consistent with the "emerging" architectural components and, if not, whether they should be regrouped so that the mapping from requirements to architectural components leads to abstraction-consistency in the architecture. Our industrial associates also use more or less a similar negotiating and verification approach between the architects and the requirements analysts, though they use a particular classification technique. For example, they categorise use cases into three levels, where lower the level of maturity of the corresponding requirement (i.e., its degree of understandability), lower the level number of the use case category, and vice versa. The more abstract use cases are left to the architects to detail in the design. Ultimately, the allocated budget will dictate where in system design the architects will "cut corners" and what exactly they will implement. Our interpretation was that the architects' decisions may not be all open for the requirements analysts to validate.

Requirements Understanding (18%): This task involves understanding specific, individual requirements, as well as the set as a whole. This step was mainly done at the start of the ADD process. The participants were given the requirements document for analysis and understanding, and for asking questions concerning any problems or ambiguities they were having with the document.

Upon closer inspection of the data, some of these misunderstandings have their roots in ambiguously expressed requirements (almost what one would consider requirements defects) – which can be fixed relatively easily. Yet, there were many cases of difficulty with the fact that functional and quality requirements are separated in the documentation yet they are integrated into a single architecture. From the technological standpoint, perhaps, this calls for innovative ways to specify, organise and manage quality and functional requirements in such a way as to help the architects probe into the documentation for *their* purposes.

Our industrial associates made a particular remark that architects and other stakeholders use requirements in quite different ways. For example, business users use requirements to communicate their needs (the *what*) and to prepare acceptance test cases, normally from mid-level use cases. Software analysts use requirements to communicate the business needs to other project resources (*who*), use them for traceability, coverage of testing against requirements (using mid-to-detailed level use cases). Architects use the requirements mainly to determine *how* the business needs are going to be met. Project managers make sure that the requirements are met by answering their own questions, concerning the schedule (*when*), resources (*who*), and processes (*how*). Developers use requirements to provide a detailed solution to all the above questions (the *what, how, who* and *when*) and, finally, the Quality Assurance agents use requirements to determine test strategies, conduct risk assessment, build test scenarios, test cases and test plan, and make sure that all answers to the questions are aligned to the product goals (the *what*). What is not clear is whether a standard requirements document hinders the understanding and interpretation by these different types of stakeholders in their different contexts and whether the requirements document needs to be customised to their specific needs.

<u>Other:</u> Of course, the findings of the study would not be complete without identifying what was *not found* to be problematic. Figure 3-1 shows that the architects encountered few RO problems in the following areas: *requirements separation*, *domain understanding*, *use-case modelling*, *constraints*, *context modelling*, *requirements coverage*, and *reasoning*. Briefly:

- Requirements separation - A fairly simple task for the architects.
- Domain understanding – This can be difficult, admittedly, but as mentioned earlier, in this study we chose a familiar domain.
- Use case modelling – Though architects did not have to model use cases, some chose to do this where they felt it was an aid and this was not a problematic task. In fact, tasks that had to do strictly with functional requirements were fairly problem-free.
- Constraints – The role of certain types of requirements as a constraint seemed to be well understood by the architects.
- Context Modelling - The main difficulty here was in separating contextual issues from what should be in the SA. Some thought aspects of this overlapped, and it could be because of the lack of distinction made in the requirements.
- Reasoning - Most architects did relate architectural decisions to the requirements. In (Bass et al., 2003), the rationale of the architecture is used to explain implications of system-wide decisions on meeting requirements and satisfying constraints, as well as the effects on the architecture if new requirements are added or existing ones changed.

3.2 RE Knowledge on Architecting

In our study, we had seven teams of architects that had requirements engineering experience and nine without. This was ascertained through background analysis at the beginning of the project. Not truly knowing beforehand the impact of the extent of RE knowledge on software architecting, we had wanted to explore the similarities and differences between these two groups in terms of RO problems during architecting.

On average, the RE-experienced teams had five fewer problems than the non-RE teams (15 vs. 20 per team, respectively, with the "Severe" problems being the same for both types of teams, and the "Mild" and "Moderate" problems contributing to the difference between the two. This overall result, however, is statistically not significant (t (14df) = 1.76131, p =0.189479). What this finding means is that *both* types of architects encountered RO problems. This is also evident from the detailed analysis shown in Figure 3-3.

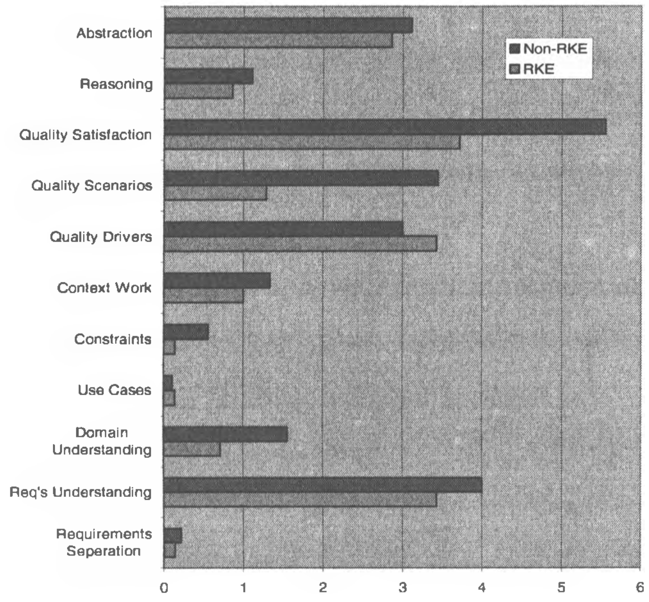


Figure 3-3. RO problems: RE vs. Non-RE teams.

As mentioned earlier, an implication of the finding from our case study could be that in the "handover" process, from requirements engineers to the architects, emphasis should be put on ensuring that the architects not only understand the requirements as documented but have comprehended them *in terms of architecting*. In this matter, our industrial associates indicated that it was not clear how the architects and the requirements analysts should integrate though senior management desired that the architects participate in the *user-centred* requirements elicitation. This was because the architects were generally capable of dealing with *technological* requirements.

3.3 Summary

The results of the study show several quality-related areas where architects had difficulties. These are: *Quality Satisfaction, Quality drivers determination, Modelling quality requirements, Abstraction and Requirements understanding.* The implications of these findings include:

- The need for further research in the way that the expression of certain types of quality requirements can be targeted at specific stakeholders in the development process.
- The need to conduct empirical studies on the modelling of, and use of, quality scenarios in industrial projects.
- A tighter integration of the RE and SA processes, for example, through "handover" processes where details of conflicts and tradeoffs analysis, and underlying assumptions and rationale from RE process are shared with and passed on to the architects.
- The need to ask or verify, during requirements engineering or architecting, whether the requirements have been documented at a level consistent with the "emerging" architectural components

There were also some areas where architects encountered few RO problems: *requirements separation*, *domain understanding*, *use-case modelling*, *constraints*, *context modelling*, *requirements coverage*, and *reasoning*.

4 Related Work

While we could not find another formal study examining RO problems during the architecting process, other researchers certainly have touched upon requirements-related issues outside RE processes. In (Kuwana and Herbsleb, 1993), for example, Kuwana and Herbsleb describe an empirical study that explored the "types of questions" various developers asked during requirements and preliminary design stages. Their findings show from two different sources that approximately 65% of the questions are requirements-oriented and approximately 35% were design-oriented. In contrast to this early work, which can be summed up as a "*pre-*

occupation" study, in our study, about 35% of the problems during architecting were requirements-oriented.

In (Nixon, 1993), Nixon describes a model for representing *performance* at requirements time to better facilitate its implementation in subsequent development phases. Our findings on the architects' difficulties with the quality attributes, *performance* and *availability* (see section 2.4.4.1 – *Quality satisfaction* subsection), support the need to further conduct this kind of research.

Although not *directly* investigating RO problems outside RE processes, there is a growing body of research aimed at creating new methods and tools that are focused on making a smoother transition from requirements to architecture. It is this type of work that the findings from studies such as ours could feed into.

One such work is Brandozzi and Perry's "Preskriptor" process (Brandozzi and Perry, 2003) which is centred on an architectural descriptor language and its associated process to systematically ensure that requirements are being satisfied. Here, our findings in the area of *Quality satisfaction* could be potentially helpful as it highlights the kind of quality satisfaction problems faced by the architects. Egyed et al. (Egyed et al., 2001) in their CBSP (Component-Bus-System and Properties) method also use an intermediate language for expressing requirements in a form that more closely relates to architecture, where requirements are identified and categorised based on various architectural and other properties. Our study supports this because we identify the need for "handover" processes between requirements and architecture (see section 2.4.4.2) where CBSP-like methods can play an important role. Liu and Easterbrook (Liu and Easterbrook, 2003) extend the CBSP method by introducing a rule-based framework that allows for requirements-architecture mappings to be automated where possible. Liu and Mei's work (Liu and Mei, 2003) is also interested in formally mapping requirements to architecture, but in their approach it is accomplished through features, where a feature is defined as "a higher-level abstraction of a set of relevant detailed software requirements, and is perceivable by users (or customers)." Thus, our findings on the architect's difficulties with *requirements understanding* (see section 2.4.4.1 – *Requirements understanding* subsection) might very well feed into this type of work.

Also, there are many other research efforts in the requirements-architecture area where our findings could possibly find a suitable home. We describe several representative examples. In (In et al., 2001), Hoh In et al. propose an eight-step framework that is based on existing RE and SA methods (WinWin and CBAM respectively) to help not only developers, but all stakeholders, to elicit, negotiate, and evaluate requirements-architecture properties while concurrently executing these processes. Nord and Soni's architecting method (Nord and Soni, 2003) deals with the identification and analysis of global factors - those that take into account more holistic issues such as the environment in which the system is built, organisation of developers, external technological solutions, "*flexibility* or *rigidity of requirements*", and more. In (Silva et al., 2003), Silva et al. describe their requirements-oriented Tropos methodology and how it defines a number of architectural patterns for various domains that take into account domain and environmental issues. Bass et al.'s ADD process (Bass et al., 2003) focuses on iteratively building architectures based on the key architectural drivers of the system. Tradeoffs emerge in the patterns between various quality attributes, and the architects and other stakeholders must negotiate a resolution to these tradeoffs (similar in principle to the Architecture Tradeoff Analysis Method (ATAM) [Kazman et al., 2000]) to finalise architectural patterns.

In all such work, there could be plausible hooks where the findings and knowledge gained from our study (and other such studies) can be fed back to improve the RE-to-SA methods, processes, and tools. We believe it is through such theory-empirical *dialogues* that the community as a whole can make efficient progress.

Yet other work is that by Rapanotti et al. (Rapanotti et al., 2004) where the concept of problem frames is extended into "architecture frames" which capture information about architectural styles and their interaction with the problem space. The benefit of this mechanism is that in introducing solution-oriented approaches early in development, one can refine problem analysis.

Finally, in (Damian and Chisan, 2006), Damian and Chisan report on a large-scale case study on the effectiveness of requirements engineering processes on other development processes such as architecting, lower-level design and implementation.

Also, they link many problems that occur later in development back to problems that originated during the requirements phase. However, the sorts of problems they have investigated are quite complementary to the ones we have investigated in our study, e.g., requirements not being properly documented and shared, relying on word-of-mouth, incompleteness, inconsistencies, etc.

5 Future Work

One purpose of an exploratory study is to lay a foundation for possible future work on the theme of the research so as to build an appropriate body of knowledge (Zave, 1997). In a sense, the exploratory study is conducted in a "bottom-up" manner, where the research question acts as a guide to collecting a wide range of data about the research topic, and the findings are discovered from the exploratory analysis of this data. In an effort to lay such a foundation, it is important to identify any emergent hypotheses or investigative questions from this research. From such hypotheses, it would then be possible to conduct, in a "top-down" manner, quantitative studies that focus on specific research issues.

A well-known Software Engineering research paradigm that can be used in a top-down manner is GQM (Goal-Question-Metric) (Basili and Rombach, 1988; Basili and Weiss, 1984). In such studies, instruments would typically need to be developed to measure the dependent variable and any other metrics required. The main purpose of conducting a "top-down" study is to statistically test the hypothesis to lend quantitative support to the topic being investigated.

From the results of our study and their implications, below we describe the following two emergent hypotheses that could be tested in future studies:

Hypothesis 1: *If the requirements engineers and software architects together model quality requirements, then the number of requirements-oriented problems during the architecting process will decrease.*

This hypothesis emerges from the finding that the architects had many problems dealing with the modelling of quality requirements during the project. In our example, the requirements engineers did not do the modelling beforehand, it was expected that the architects would model the scenarios based on the key quality requirements. Evidently, the architects had many difficulties with this which centered around two main issues: (a) identifying the purpose and benefit of modelling quality scenarios; and (b) conceptualising how the quality scenarios would fit into the various levels of abstraction in the architecture. We believe that if the requirements engineers and software architects model the quality requirements together, this would not only give the architects a better understanding of the specific purpose of each model, but also an increased knowledge of the key quality drivers for the system. It is also likely that the resultant quality models would fit better with the existing architecture due to knowledge transfer from the architects to the requirements engineers.

A related point is that this hypothesis testing could also demonstrate the cost/benefit of modelling quality. Are such models worth building? If so, how many, or how rigorous, should we strive to build for each system before the time and cost outweigh the benefits?

To test this hypothesis, we would need to measure and compare the number of requirements-oriented problems that occur in two different groups of architects. One of these groups would conduct the quality requirements modelling with the requirements engineers; whereas, the other group of architects would model the drivers without the involvement of the requirements engineers. In this hypothesis, the independent variable would be the quality requirements modelling, and the dependent variable is the number of requirements oriented problems.

<u>Hypothesis 2</u>: *If adequate background information about the requirements (such as, rationale, assumptions, priority, etc.) is given to, or shared with, the software architects (either through a handover process or formal documentation) then fewer requirements-oriented problems will be encountered by the architects.*

By intuition, this hypothesis would seem to be true. However, there is a lack of empirical knowledge pertaining to this hypothesis and the overall usefulness of full background requirements information that should be made available to the architects. In many organisations it would be quite costly to fully document their requirements, or to add extra processes that involve the requirements engineers and the software architects. Testing this hypothesis would bring quantitative data in either justifying this extra "work", or refuting it.

To test this hypothesis, we would measure the number and severity of requirements-oriented problems that occurred in the conduct of a software architecting project with two groups of architects. One of these groups would not be given access to the full background information about the requirements (rationale, assumptions, etc.); the other group of architects would be given the full documentation. In this hypothesis, the independent variable is the requirements background information, and the dependent variable is the number and severity of requirements-oriented problems.

Aside from the hypotheses, more in-depth analysis can possibly be carried out in such areas as Requirements Understanding (see Figure 3-2). We see here that severe, moderate and mild problems constituted 8%, 23% and 68%, respectively, of the total requirements understanding problems. However, in this study we did not break these down further into, for example, user-oriented and technological requirements. Such decomposition could give further insight into specific aids that could be devised for the architects.

6 Conclusions

Based on our analysis of the requirements literature, an insignificant amount of research has been carried out on requirements *outside* the requirements engineering (RE) process. By conducting empirical studies of requirements in non-RE processes, much feedback can be gathered which can be invaluable for improving both requirements and RE technologies. In this paper, we describe a case study of requirements-oriented (RO) problems experienced by sixteen teams architecting the same banking application. The study found that the key RO problems, of varying severity, were:

- *Quality Satisfaction* (22%)
- *Quality drivers determination* (15%)
- *Modelling quality requirements (scenarios)* (12%)
- *Abstraction* (14%)
- *Requirements understanding* (18%)

In addition, the study found that about a third of all problems were RO problems, which should thus be a source of concern to the RE community in that there may be ways to reduce RO problems in non-RE processes. However, the study found relatively few RO problems in the areas: *requirements separation*, *domain understanding*, *use-case modelling*, *constraints*, *context modelling*, *requirements coverage*, and *reasoning*. The paper also describes some implications of the findings for the RE field, particularly in the areas of: expression of quality requirements for different stakeholders; empirical studies on quality scenarios; tighter integration of RE and SA processes; and requirements to architecture mapping. There was much concurrence of our findings with expert opinion from a large insurance company.

Since this was only one case study in a particular context, and despite our validation through industrial associates, we caution the liberal use of these results in other contexts. It would be ideal to first conduct more such studies. However, such a case study in industry is non-trivial. It would involve selection of an appropriate architecting project, which would clearly need a "buy-in" from the project staff. Also, appropriate project deliberations (e.g., discussion of RO problem areas) would need to be gathered, through project meeting logs or data gathering templates. Practitioners can gather their own data (as in Action Research [Mason, 1996]) or this could involve a researcher as observer (as in Ethnographic studies [Hancock, 2002]). If the study captures data from a specific moment in the project then this would be a "case" study; if it captures data over a long period of time then this would be a "longitudinal" study. Due to continuous development cycles in industry, the latter types of studies are plausible or even desirable for quality findings. The aspect of separating RE vs. non-RE project-staff in industrial projects, however, could be extremely difficult because of their accumulated experience over long periods of time.

Despite the described limitations, it does *not* diminish the importance of our results; in fact, all the more, it lays a foundation for future analogous studies so that, one day, meta-analysis can be carried out over accumulated results. In this respect, a relatively new area of research that unfolds with this case study is that of conducting requirements studies in all sorts of *non*-RE processes, not only the architecting process considered in our study, so that much needed feedback can be passed on to the RE technologists.

ACKNOWLEDGEMENTS

A number of researchers had given to us quite a significant amount of their invaluable time, knowledge and encouragement in the research procedures, such as data transcribing and validation, categories validation, data analysis validation and more. We are much indebted to them. We are thankful to our industrial associates for their precious time and professional opinions on our findings and for drawing analogies in their industrial context. We would also like to thank deeply the anonymous reviewers for their valuable comments and feedback.

References

Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; Wood, W., 2003. Quality Attribute Workshops (QAWs), Third Edition. SEI Technical Report, 2003.

Basili, V.R. and Rombach, H.D., 1988. The TAME project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering, SE-14(6), pp. 758-73.

Basili, V.R. and Weiss, D., 1984. A methodology for collecting valid software engineering data. IEEE Transactions on Software Engineering, SE-10(6), pp. 728-38.

Bass, L.; Clements, P.; and Kazman, R., 2003. Software Architecture in Practice, 2nd edition. Addison-Wesley.

Berander, P., 2004. Using Students as Subjects in Requirements Prioritization. Proceedings 7th International Conference onEmpirical Assessment & Evaluation in Software Engineering, pp. 95-102.

Brandozzi, M. and Perry, D. E., 2003. From Goal-Oriented Requirements to Architectural Prescriptions: The Preskriptor Process. STRAW workshop.

Creswell, J. W., 2003. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Thousand Oaks, CA: Sage Publications.

Damian, D. and Chisan, J., 2006. An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. Transactions on Software Engineering, 32(7), 433-453.

Egyed, A.; Grunbacher, P.; Medvidovic, N., 2001. Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach. STRAW workshop.

Ferrari, R. and Madhavji, N. H., 2007. Impact of Requirements Knowledge and Experience on Software Architecting: An Empirical Study. Sixth working IEEE/IFIP Conference on Software Architecture, India.

Ferrari, R., and Madhavji, N. H., 2008. Architecting-Problems Rooted in Requirements. Special Journal Issue of Information and Software Technology on Best Papers from REFSQ'05 and '06, Volume 50, Issue 1-2, January 2008, pp. 53-66.

Finkelstein, A., 2000. The Future of Software Engineering. ICSE 2000 Proceedings.

Graham, D., 2006. Testing to improve requirements – is it mission impossible?, 14th Requirements Engineering Conference (RE06), Minneapolis, USA.

Guion, L., 2002. Triangulation: Establishing the Validity of Qualitative Studies. University of Florida Extension: Institute of Food and Agricultural Sciences.

Hancock, B., 2002. An Introduction to Qualitative Research. Trent Focus for Research and Development in Primary Health Care.

Host, M.; Regnell, B.; Wohlin, C., 2000. Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. Empirical Software Engineering, pp. 201-214.

In, H.; Kazman R.; and Olson, D., 2001. From Requirements Negotiation to Software Architectural Decisions. STRAW workshop.

Kazman, R.; Klein, M.; Clements, P., 2000. ATAM: Method for Architecture Evaluation. SEI technical report.

Kuwana, E., Herbsleb, J. D., 1993. Representing Knowledge in Requirements Engineering: An Empirical Study on what Software Engineers Need to Know. Proceedings of IEEE International Symposium on Requirements Engineering, pp. 273-276.

Liu, D. and Mei, H., 2003. Mapping requirements to software architecture by feature-orientation. STRAW workshop.

Liu, WenQian and Easterbrook, S., 2003. Eliciting Architectural Decisions from Requirements using a Rule-based Framework", STRAW workshop.

Madhavji, N.H. and Perry, D., 2004. From Software Requirements to Architectures, CASCON workshop, Toronto, 5th October.

Mason, J., 1996. Qualitative Researching. SAGE Publishing Ltd, London.

Metze, L., 2001. Pride Technical Report: a Reliability Study. Western Kentucky University. This report is available at *http://www.pridesurveys.com/supportfiles/tr9946.pdf*. (Last access 2006)

Nixon, B, 1993. Dealing with Performance Requirements During the Development of Information Systems. Proceedings of IEEE International Symposium on Requirements Engineering, pp. 42-49.

Nord, R. L. and Soni, D., 2003. Experience with Global Analysis: A Practical Method for Analyzing Factors that Influence Software Architectures. STRAW workshop.

Nuseibeh, B., 2001. Weaving the Software Development Process Between Requirements and Architectures. STRAW workshop.

Ramesh, B. and Jarke, M., 2001. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, pp. 58-93.

Rapanotti, L., Hall, J., Jackson, M. and Nuseibeh, B.,, 2004. Architecture-driven problem decomposition. In Proceedings of 12th IEEE International Conference on Requirements Engineering (RE 2004), pages 80–89. IEEE Computer Society.

Runeson, P., 2003. Using Students as Experiment Subjects – An Analysis on Graduate and Freshman Student Data. EASE'03 – Proceedings 7th International Conference on Empirical Assessment & Evaluation in Software Engineering.

Silva, T.; Castro, J.; Mylopolous, J., 2003. Detailing Architectural Design in the Tropos Methodology. STRAW workshop.

Software Requirements to Architectures Workshop (STRAW), ICSE workshop, 2001 and 2003.

Sommerville, I and Sawyer, P., 2000. Requirements Engineering: A Good Practice Guide, John Wiley & Sons Ltd.

Thelin, Thomas, 2004. Team-based fault content estimation in the software inspection process. 26th International Conference on Software Engineering (ICSE 2004), pp. 263-272.

Zave, P., 1997. Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys, Vol. 29, No. 4, pp.315-321.

Chapter 4

Characteristics of New Requirements in Presence/Absence of Existing Systems Architecture[22]

1 Introduction

While much research attention has been paid to transitioning from requirements to system architectures (SA)[23] (STRAW, 2001 and 2003), relatively little attention has been paid to how new requirements are affected by an existing SA. Indeed, it was over a decade ago, in a panel session (Shekeran, 1994), when several concerns and thoughts expressed the need to consider SA during requirements engineering (RE), for example: "We still do not have a clear understanding of the role of software architecture in requirements engineering" (Shekeran, 1994); "Software architecture must be considered during requirements engineering to ensure that requirements are valid, complete, consistent, feasible, etc." (Nuseibeh and Easterbrook, 2000). Also, SWEBOK (IEEE SWEBOK, 2004) – the software engineering body of knowledge – for example, does not describe any practices to deal with this issue.

Thus, to explore this matter further, given its thin baseline, we first conducted a survey (Miller et al., 2008) of 17 experienced RE and SA researchers and practitioners from North America and Europe. We found that the average rating of the importance of considering existing architecture when engineering new requirements was 4.5 (on a 5-point Likert-scale) – implying that the respondents strongly agreed with this concept. Despite this, several respondents noted in the qualitative part of the survey that, in actual practice, many organizations neglect this

[22] A version of this chapter has been published in (Ferrari et al., 2010). The final publication is available at www.springerlink.com.
[23] For the rest of the paper, the acronym SA refers to System (or Software) Architecture as a software artefact.

consideration, or perform analysis only on existing high-level features (i.e., the requirements) of the current system, and not on the system's architecture.

Although there is curiosity in the RE community about the impact of SA on RE, and that there is a dichotomy between theory and practice, to our knowledge no scientific studies have ever been conducted to investigate this issue. Thus, we still do not truly know the "characteristics" of the newly elicited requirements in terms of how they are affected by the presence or absence of a SA in the RE process.

For example, firstly, a general question is: to what extent are new requirements affected by the existing SA? Also, the extent to which they are affected, what are the characteristics of this effect? For instance, to what degree are the requirements "user-needs" focused, "technological-needs" focused, or "architecturally focused"? Etc. There are a number of such questions to which the RE research and practice community has no specific answers.

Having a grounded body of knowledge on these issues could benefit RE practice in a number of ways. For example, it could help in determining:

- when in the RE process one should examine the SA to ensure fitness of the new requirements;
- when in a product's lifecycle it is advantageous *not* to be influenced by the existing SA;
- the extent to which the system's requirements are misaligned with the business goals; and
- the requirements characteristics that should be tweaked in order to bring them back in line with the business goals.

Ultimately, such investigations are aimed at increasing the general quality and relevance of the system, improving RE processes, and at improving business efficiency and profitability.

Motivated by these issues, we conducted an exploratory, controlled, study to characterise the differences in the newly elicited requirements in the presence or absence of the SA. The study involved two types of groups. One type of group (the SA-group) received the SA of an existing (banking) system; whereas, the second type of group (non-SA group) did not receive the SA of this system. Both types of groups received the same initial requirements for this system and they were both asked to enhance the system's requirements given the same problem description (or project goals).

This paper describes this empirical study and its findings in quantitative terms. For example, specific biases of various requirements characteristics in the presence/absence of the existing SA are given and interpreted. These findings constitute new knowledge and are the chief contribution of the paper. The paper also describes the implications of the findings for both RE practice (e.g., RE process engineering, post-requirements analysis, traceability management) and RE research (e.g., seven emergent hypotheses, RE tools).

This paper is a significantly enhanced version of (Miller et al., 2009). The additions to this paper include:

- The investigation of a new research question regarding specific aspects of the SA that affected the requirements.
- A significantly expanded related work section.
- More information given on the data analysis conducted.
- Extended implication section including new hypotheses for further empirical studies.
- Elaboration of empirical study procedures employed.
- Appendix describing data collection instrument used.

The rest of the paper is structured as follows: Section 2 describes related work; Section 3 describes the empirical study; Section 4 analyses the data, presents the results and makes interpretations. Section 5 discusses the implications of the findings; and Section 6 concludes the paper.

2 Related Work

In this section on related work, we focus on two key aspects: (i) observations, commentary and empirical work on the role of SA in RE, and (ii) recent technological-based research on requirements evolution. Subsection 2.3 concludes with a reflection on the current state of research described in subsections 2.1 and 2.2. Other aspects that are related (for example, technology to transition from RE to SA, or empirical studies focused on requirements-oriented issues while architecting) are omitted here because they are not as relevant as the two aspects identified above. The reader can refer to (Ferrari and Madhavji, 2008) for a thorough discussion of related works focused on the transition from RE to SA.

2.1 Role of SA in RE

As early as 1994, a panel session at a RE conference was held to deliberate on the role of SA in RE (Shekaran, 1994). This marks perhaps the first attempt by the RE research community to recognise this relationship[24]. The consensus in this panel session was that this relationship is an important one but was little understood.

In this same session, Jackson (Jackson, 1994) gave four key reasons as to why RE and architecting are best treated as interweaving processes. First, RE can be "very tricky" in that, often, it can be simpler to start by building the system right away, even if only in outline. Second, evaluating possible system designs early can help gain an important understanding of which requirements might not be feasible, saving time and money. Third, requirements can sometimes be reasonably embedded in system design, eliminating the need for formal specification during RE. Finally, there is evidence that successful developers are those who are able to move relatively more freely between stages (i.e., RE, architecting, design, testing, etc.) within the development cycle.

Shortly thereafter, in 1995 (El Emam and Madhavji, 1995), El-Emam and Madhavji found four factors for RE success in information systems that deal with architecture or the system (the first being relevant for this study): the *adequacy of diagnosis* of the existing system (which includes SA); the *fit between the architecture and the way users work;* the *fit between the recommended requirements solution and the strategic orientation of the organization;* and the *fit between the recommended solution and the technical orientation of the organization.*

Subsequently, hints can be found in the pedagogical literature (Kotonya and Sommerville, 1998) promoting the need to consider the existing system in the RE

[24] Related workshops, such as STRAW '01 and '03 (STRAW, 2001 and 2003) focused mainly on transitioning from RE to SA and not on the role of SA in RE.

process. More recently, in 2000, Nuseibeh and Easterbrook (Nuseibeh and Easterbrook, 2000) stated that we needed a "better understanding of the impact of software architectural choices on the prioritization and evolution of requirements." In (Nuseibeh, 2001), Nuseibeh describes the "twin-peaks" model, which captures the iterative relationship between RE and architecting. An important aspect of this model is that the architecting process can and should feed back into the RE process (as well as vice versa).

In a recent study (Miller et al., 2008), Miller, Ferrari and Madhavji investigated the different types of effects a SA has on requirements decisions. They identify and quantify four principal ways in which a previous architecture can affect evolving requirements work, i.e., as an enabler (30%), as a constraint (25%), as an influence (6%) or the null case (39%). This means that approximately 60% of the decisions were affected by the architecture, highlighting the impact an existing architecture has on RE.

While these are some of the key works highlighting the role of SA in RE, the body of knowledge on this topic is fairly thin overall and has basically remained static.

2.2 Requirements Evolution

An area of research that is related to our work is requirements evolution, in particular from the viewpoint of methods, notations, and tools development. In the following subsection we highlight recent research in this area from prominent RE literature sources. Because our study is focused on both the *absence* and *presence* of SA in RE, we include research that does not, explicitly or implicitly, consider the existing SA in requirements evolution.

In (Vilella and Doer, 2008), the authors present a method for requirements engineers and project managers to perform software evolution in the domain of embedded systems. The method's primary purpose is to aid in systematic reasoning on the identification of volatile requirements, and planning changes to the architecture. The method is composed of four phases. The first phase is preparation for volatility analysis and is meant to establish the timeframe restricting the current volatility analysis and identifying the types of components that will be involved in the changes. The second phase is environmental change anticipation, where the primary tasks are to identify and characterize changes that may occur in the system's environment within the identified timeframe. Specifically, the analyst needs to identify actors, roles, external events and environmental facts that could cause changes. The third phase is the actual change impact analysis, which is composed of identifying the adaptation needs, such as identifying features to be affected and estimating their business impact. The result of this phase is a prioritized list of adaption needs that should be included for implementation. The final phase is the product evolution planning, where the analysts establish when and how the previously high priority adaptation needs are to be introduced into the system. The result of the method is a plan for product evolution based on the high-priority adaptation needs.

In (Etien and Salinesi, RE 2005), the authors present a framework that defines challenges for RE caused by co-evolution, and also show which and how existing requirements technologies address the identified challenges. Their framework is structured around five dimensions which each correspond to a RE-related issue regarding co-evolution. These dimensions were determined from their experience in three industrial evolution projects. To summarize, the five dimensions are: (1) understanding the consistency relationship between RE-related artefacts and other co-evolving artefacts from outside RE (e.g., design, testing, code, etc.), (2), formalizing notations to express evolution requirements, (3), elicitation of evolution requirements, (4), propagating identified changes to processes outside RE, and, (5), verifying the relationships between the proposed changed system entities. The authors conclude that no particular existing technology addresses all of the above dimensions of co-evolution, and therefore that a research gap exists in this area.

In (John et al., RE 2002), the authors propose the use of a domain analysis approach to identify and document current and future requirements in an application domain. The author's primary motivation for this approach is that defining a long-term strategy for software product evolution is an extremely difficult task because the requirements and future trends must be anticipated in advance. They argue that a domain analysis technique can be used for this anticipation of future requirements, while recognizing the problem that ongoing domain analysis for evolutionary purposes can be costly (in terms of time and cost). Thus, they propose to use an instantiation of the PuLSE-CDA (Customized domain analysis) method (Bayer et al., 2000), which aims to overcome this problem by systematically coordinating domain analysis effort with necessary product evolution activities. The primary goal to facilitate a cost-effective approach is to only model a sub-domain where only the key future changes are modeled, in order to reduce excess modeling of irrelevant information. In short, the main steps are: (1) to analyze existing change requests from maintenance and marketing, in addition to analyzing existing application domain knowledge, which provides an initial list of sub-domain candidates, (2), map the identified candidates to logical software components; (3), Model and refine each logical component's relevant data attributes and processes in which the components are involved. The output of this method is a map of inter-related domain models (and logical components) that are a subset of the overall application domain. Each component can then be implemented and integrated with the existing system.

In (Breitman and Sampaoi, 2001), the authors investigate requirements evolution from the perspective of scenarios. Specifically, they derive a scenario evolution taxonomy from the investigation of twelve case studies spanning over 200 scenarios; each of these studies comprised the analysis of a software project during its evolutionary phase. The authors state, based on the findings from the case studies, that the main challenges in scenario evolution are in understanding and managing the relationships between scenarios; an individual scenario can often be related to many other scenarios and in the projects examined in the case study, there was minimal technological support for this problem. The resultant scenario evolution taxonomy then describes the classification and formal heuristics for semi-automated detection of scenario relationships, as well as an initial suggestion for a formal notation that can be used for scenario relationship representations.

In (Ferreira et al., 2009), the authors propose a simulation model to help project managers and requirements analysts understand how requirements volatility impacts a given software development project. The model is built on results from an empirical survey administered to software project managers and developers, where more than 50 parameters (such as number of requirements change requests per release, requirements defects detection rate during design, percentage of perceived job size increase due to requirements change) were derived from the survey data. Based on this theoretical model, the authors designed a software simulator that can be used by developers to input project parameters that are related to requirements volatility and determine the potential impact for a given project of changing requirements. The simulator was used on two industrial projects to explore the relationship between requirements volatility and its impact on software projects. The results of these case studies show that there were significant simulated cost, schedule and quality impacts due to requirements volatility.

In (Rolland et al., 2004), the authors tackle the problem of requirements evolution with a formal requirements specification modeling approach and tool. Currently, this approach works on specifications modeled using i* (Yu, 1997). The aim of the approach is the precise definition of the change requirements, and the approach does this by facilitating the modeling of specific gaps between the current requirements specification and the target specification. The approach uses a generic gap typology where each gap is associated to a predefined type of requirements change (such as add actor, remove feature, etc.), and these are then associated with gap operators which perform the actual transformations in the i* model. The approach and tool was validated, and the authors estimate that approximately 50% time was saved eliciting change requirements using this approach vs. a manual approach.

In (Burgess et al., REJ 2001), the author's raise the problem that in large software projects, the number of changes and enhancements requested for inclusion in the next release often exceeds the resources available to implement those changes. Therefore, technological support is needed to incorporate the multitude of factors (such as approval for finance for the change, development time, human expertise required, etc.) that influence these possible changes into an improved set of information for the purpose of facilitating better decision-making. The authors propose the use of influence diagrams, which are an extension to Bayesian nets (Castillo et al., 1997), to formalize the combining of the different change factors to address the requirements change problem. Influence diagrams, as argued by the authors, are suitable because they model both decision-making trees along with random chance events. The combination of these two dimensions adequately covers both fixed and volatile project factors that can influence decisions to implement certain requirements changes.

2.3 Reflection on Research

Having discussed the current knowledge pertaining to the role of an SA in RE and requirements evolution, in this subsection, we reflect on the current state of research in these areas. As discussed In Section 2.1, as early as 1994, researchers discussed the importance of the role of an SA in RE (Shekeran, 1994). A few other works have commented on this issue since then (El Emam and Madhavji, 1995; Nuseibeh and Easterbrook, 2000; Miller et al., 2008). However, beyond these works there has been, to our knowledge, little research conducted in the area of the role of an SA in RE. Despite the sparse research in this specific issue, there is a wide range of technological-based research done in the area of requirements evolution, as described in Section 2.2, which are meant to improve the RE process in the context of an evolving system. However, the work is often focused solely on the RE process; downstream activities such as architecting, coding, testing, etc. are treated as black-box processes and there is thus a lack of explicit recognition of the interaction of RE and SA as highlighted as being important, for example, in Nuseibeh's Twin Peaks Model (Nuseibeh, 2001). Furthermore, there is a lack of empirical evidence regarding the different requirements characteristics, and how these characteristics are impacted by the presence or absence of an SA during systems evolution. The empirical study presented in this paper is meant to present detailed quantitative findings on the impact of an SA on requirements characteristics, which can then be fed back into technological research as described in Section 2.2. .

Though the importance of conducting empirical studies in software engineering (SE) has been recognised (Tichy et al., 1995; Wieringa and Heerkens, 2006), Shaw's analysis (Shaw, 2003) of research papers submitted at a prominent 2002 SE conference suggests that only 12% were submitted in the category of "Design, evaluation, or analysis of a particular instance" and 0% in the category of "Feasibility study or exploration". In (Ferrari and Madhavji, 2008), we presented our own analysis of published papers. In the fields of RE and SA, since the year 2000, only approximately 15% of the published papers were in the above-mentioned categories, suggesting that studies such as the work described in this paper are currently rather rare. Our work is meant to help in filling this research gap.

3 The Study

We now describe the core parts of the study. Section 3.1 describes the research paradigm used, GQM (Basili and Weiss, 1984), to state the goal, questions and metrics for this study. Section 3.2 describes the study design. Section 3.3 describes the study hypothesis. Section 3.4 describes the participants. Section 3.5 describes the RE project. Section 3.6 describes the data collection procedures. Finally, Section 3.7 describes threats to the study.

3.1 Goal, Research Questions and Metrics

This study followed the Goal-Question-Metric (GQM) paradigm, which helps in ensuring that measurements taken in the study are aimed at answering specific research questions which, in turn, help in achieving the overall goal of the study (Basili and Weiss, 1984).

The overall goal of this investigation was:

To better understand the characteristics of requirements elicited in the absence or presence of a SA.

In order to obtain a quantitative insight into the elicited requirements, we decomposed the notion of a requirement into specific, measurable, characteristics. Table 4-1 defines these characteristics, which are rooted in three sources: those which are prominent in the literature (such as requirement type); those which would intuitively be of interest to industry (such as cost); and those which relate to an architecture (such as architectural relevance (Bass et al., 2003). Five researchers subsequently validated these characteristics, and their associated metrics, as an acceptable set of variables for the study.

(1) *Focus on Cost* - The degree to which the cost factor, concerning the system's content, is prominent in the requirement.
(2) *Focus on Time* – The degree to which the development time factor, concerning the system's content, is prominent in the requirement.
(3) *Focus on Quality* - The degree to which the quality factor, concerning the system's content, is prominent in the requirement.

Note: a requirement can be prominent in one or more of the above three characteristics.

(4) *Focus on user's needs* -The degree to which the requirement is focused on the needs of the end-users. End-user issues include: different ways of accessing the system, end-user features provided, usability requirements, etc.
(5) *Focus on client's needs* - The degree to which the requirement is focused on the needs of the client (i.e., the needs of the organisation itself.) Note the difference from (4) above.
(6) *Focus on technological needs* - The degree to which the requirement is focused on technological needs. Technological issues include: the "back end" server, choice of algorithms and data types, interface specifications, communication protocols, data access mechanisms, etc. that are important in terms of ensuring that the system will be technically sound.
(7) *Testability* - The degree to which it can be shown that a requirement can be tested against.
(8) *Implementability* - The level of effort required to implement a requirement.
(9) *Importance* - The degree to which the success of the system depends on the implementation of a requirement.
(10) *Architectural relevance* - The degree to which the requirement will have an impact on the architecture, e.g., architectural-driver. Note: not all "important" requirements are architecturally relevant, e.g., a common but basic requirement.

Scale for characteristics (1-10) above: These were measured using a 7-point Likert scale. The scale was the same for all characteristics and is defined as follows: 7 – Very high, 6 – High, 5 – Moderately high, 4 – Neither high nor low, 3 – Moderately low, 2 – Low, 1 – Very low.

(11) *Level of abstraction* - The breadth of impact of a requirement. Does the requirement affect a module, a component, an entire sub-system, etc.

Scale: This was measured with a 7-point ordinal scale; this scale was used because there is an ordering in the levels of abstraction (high to low for example), but this ordering does not denote specific, discrete values, with equal intervals between them (Pett, 1997).

(12) *Type of requirement* - This categorizes the requirement into functional, non-functional, and business quality, where there can be more than one category for non-functional and business qualities. Examples of categories in this scale include *functional, standards, legal, performance,* and *safety.*

Table 4-1. Requirement characteristics

Linking the overall goal and the characteristics described in Table 4-1 is the following question aimed at achieving the goal:

Q1: Which requirement characteristics were affected, and to what extent, by the presence or absence of the SA?

Our objective now is to determine whether or not the SA (an *independent* variable) has an impact on the requirements characteristics (the *dependent* variables). We can accomplish this by comparing the requirements sets elicited by two different types of study groups, one group which is comprised of teams that have access to the SA while doing RE (the SA-groups), and the other set of teams do not have access to the SA (the non-SA-groups). To complement and probe deeper into the findings from Q1, we raise the following research question.

Q2: Which specific aspects of the SA affected the requirements?

In this question, we are examining the specific aspects of the architecture and how they affected the requirements characteristics that exhibited significant differences (i.e., findings from Q1). To investigate this question the requirements analysts teams, during the RE process, had to explicitly state when an architectural aspect affected their RE work. We then take the intersection between the identified affected architectural aspects and the requirements characteristics with significant differences from Q1.

3.2 Hypothesis

Because this study was undertaken without a significant underlying theory (on how requirements characteristics are affected by systems architectures) on which to build a priori hypotheses, this study is best described as an exploratory study (Rao, 1997). Contrary to a dogmatic viewpoint, hypothesis testing *can* be done in exploratory studies but is not meant to confirm existing scientific theory as in a purely experimental design. Rather, the results of the hypothesis testing here provide initial indications on which future experimental research can be conducted (Rao, 1997). As a result, the only hypotheses that will be discussed in this paper are the "null hypotheses" which are necessary for null hypothesis statistical testing (NHST). The null hypothesis for a given metric M, where M is a requirement characteristic described in Table 4-1, can be generally stated as:

H_0: the presence of a SA has no impact on M.

If NHST leads to the rejection of this hypothesis for any metric M then we can say that "characteristic M is affected by the SA".

3.3 Study Design

This is a *post-test* only control group design experiment (Johnson and Christensan, 2003). This type of design involves administering a treatment (i.e., SA documentation) to one type of groups (in this case, the SA-groups), with observations taken only at the end following the treatment. These observations are contrasted against the non-SA-groups that did not receive the treatment. It is from this contrast that the results of this study are drawn. A visual depiction of this design is given below; where the O represents observation, X represents treatment, and R represents random assignment. This study design was used because it falls in the category of *strong designs* (Johnson and Christensan, 2003), and alleviates many internal validity threats in a multiple group design.

SA-Groups	R -------X ------------- O
Non-SA Groups	R ------------------------O

The specific type of experimental design that our study falls under is a nested-ANOVA design (Rao, 1997). This design is used when there is one measurement variable (i.e., requirement characteristic), and two or more nominal variables (i.e., categorical variables). In our study, there are two nominal variables: (1), the type of study group (SA-Groups vs. non-SA groups) and (2), the different requirements teams. These nominal variables are nested, meaning that each requirements team belongs to only one category of the higher-level nominal variable (i.e., the SA-Groups vs. non-SA groups). We used this design because although our *unit of observation* is the requirements teams, the *unit of analysis* was the individual requirements since we were interested in the differences in the requirements characteristics, and not differences in teams. The subsequent analysis (see Section 4.1) supports this study design and reconciles the difference between *the unit of analysis* and *unit of observation*.

3.4 Participants

We used convenience sampling (Johnson and Christensan, 2003) to involve 25 final-year RE students in the study. In order to conduct the study involving students, we received consent from the ethics board at The University of Western Ontario. The threat of using students as participants is discussed in the external threats to validity section (Section 3.6.2). The participants were randomly assigned to groups of two with one group of 3, making a total of 12 groups. The groups were then randomly divided into two types: the so-called "architecture (SA) groups" and "non-architecture (non-SA) groups".

To ensure that the participants had sufficient knowledge to conduct the project, they were given theory knowledge in RE and two pre-requisite requirements projects prior to the project to learn and familiarize themselves with RE practices such as elicitation, analysis, negotiation, validation, and prioritisation. The assessment of the pre-requisite projects and RE knowledge indicated a satisfactory level of attainment to conduct the investigative project. Subsequent to this, SA learning sessions were given to the SA groups, so that they could perform architectural analysis required for the project. These sessions focused on understanding architectural documentation and the ADD/ATAM methods for architecture design and assessment (Bass et al., 2003).

3.5 The RE Project

Each of the 12 groups was given the same set of requirements elicitation tasks for upgrading a software infrastructure for a fictitious bank:

1. To add support for *Interac* service to the existing system.
2. To create a new wireless banking application which would allow customers to carry out basic banking transactions through their cell phones or PDAs.
3. To reduce the operational cost of the telephone banking system.
4. To increase modifiability in the web banking system.

These tasks were chosen since they constituted a sizeable and complex RE project that would still be feasible within the constraints of a University course. We held numerous peer-review sessions with a total of six experts to validate these four tasks with respect to their appropriateness in giving a project that met both pedagogical and study needs.

Both types of groups, SA and non-SA, were given the requirements for the existing system. These pre-existing requirements were baselined from a previous project (Ferrari and Madhavji, 2008). The requirements elicitation process and techniques followed are described in (Kotonya and Sommerville, 1998).

Also baselined was the pre-existing architectural document (developed using the ADD method (Bass et al., 2003)) from the same previous project (Ferrari and Madhavji, 2008), given to each of the SA-groups only. This document included architectural information such as: numerous different tactics, quality attribute scenarios, decomposition views, user/layer views, class views, component and connector views, deployment views, work assignment views, sequence diagrams and state charts. The RE project, including logging of the elicited requirements, took two months to complete.

3.6 Data Collection

The data collected for analysis were the ratings for the requirements-characteristics (defined in Table 4-1) for the elicited requirements (to answer Q1 – see Section 3.1), and the list of architectural aspects that affected the SA-group during the RE process (to answer Q2 – see Section 3.1). In the following two subsections, we discuss these two disparate sources of data.

3.6.1 Requirement Characteristics Ratings

The primary source of data is the requirements ratings for the requirements-characteristics (defined in Table 4-1), where an instrument was designed for this purpose (see Appendix B). Three external researchers rated the requirements using this instrument. This process involved examining the title, description and rationale for each requirement and giving a rating on the appropriate scale of each and every characteristic.

Since the data was subjective and different raters could measure constructs in different ways, an inter-rater agreement method from (Fusaro et al., 1997) was used to establish rating reliability. Basically, the ratings for each characteristic for each requirement were assigned to two researchers. The two researchers independently performed the ratings. Following this, if necessary, the researchers harmonized their ratings to finalize a rating. In particular, if there was a difference of more than one in the ratings, a third researcher also performed the rating to reconcile the difference. All ratings were conducted blindly, i.e., without knowledge of which group authored which requirements. The researchers used for the rating procedure all had 3-10 years experience in RE, and the minimum academic level was a PhD candidate. Prior to the ratings collected for this study, a pilot rating session was conducted on a subset of the requirements with all researchers involved, the purpose of which was to train the raters on the rating procedure. Table 4-2 shows the ratings of two example requirements, R1 and R2.

R1: *When performing an Interac transaction, if the primary server is busy it will send a 'Server busy' response signal followed by a secondary server IP. The client machine should then have to re-establish connection with the secondary server and perform a second request. This results in less demand on the primary server.*

R2: The customer shall be able to add or remove companies from their profile to which bills are being paid using their mobile application (cell-phones, PDA's, etc.).

Requirement id	Focus on Cost	Focus on Time	Focus on Quality	Focus on user's needs	Focus on client's needs	Focus on tech needs	Testability	Implementability	Importance	Arch. Relevance
R1	4, 4, ***4***	4, 4, ***4***	4, 4, ***4***	2, 1, ***1.5***	1, 1, ***1***	7, 7, *7*	7, 6, ***6.5***	7, 7, 7	5, 4, ***4.5***	2, 3, ***2.5***
R2	4, 3, ***3.5***	4, 3, ***3.5***	4, 4, ***4***	7, 6, ***6.5***	1, 2, ***1.5***	1, 1, ***1***	7, 7, 7	4, 5, ***4.5***	5, 4, ***4.5***	4, 4, ***4***

The first two numbers in each column represent ratings given to the requirement for the given characteristic by two independent raters (see Table 4-1 for explanation of scale). The third value in bold italic is the final, agreed upon value. One example interpretation of this instance of ratings is that R1 is high on technological-focus (7 out of 7) but low on user-focus (1.5 out of 7) and R2 is high on user-focus (6.5 out of 7) but low on technological focus (1 out of 7). Likewise, interpretations can be made about other characteristics and their ratings. Because the scale for the characteristics Abstraction and Requirement Type are different, we omit them in the example for simplicity.

Table 4-2. Sample requirements ratings.

3.6.2 Architectural Aspects

We used a tool from (Miller et al., 2008) that had the dual purpose of supporting the subjects' project and of recording relevant data for this study. For example, the tool maintained a pervasive list of both the original requirements (as well as their evolution) and new requirements introduced by each team. Also, the tool recorded data from the SA-groups (only) about the specific parts of the architecture that had an impact on the requirements. To help ensure the quality of the data gathered in this tool, regular meetings with each project team were held to clear up issues and to monitor the progress.

3.7 Threats to Validity

We classify threats into those *internal* and those *external* to the project, as well as *construct* and *conclusion* validity. We focus here only on those considered relevant to our study. Description of other types of threats can be found in (Johnson and Christensan, 2003).

3.7.1 Internal Validity

Internal validity deals with whether we can infer that a relationship between two variables is causal, and not due to any confounding factors (Johnson and Christensan, 2003). There are numerous specific types of internal validity threats (Johnson and Christensan, 2003), we discuss here only the threats that applied to our study and the procedures we employed to contain the threat.

Differential selection: This is when possible characteristics of the subjects may, by chance, differ between the two types of groups and possibly affect the quality of the data. In our study, such a characteristic is the participants' SE educational and industrial-experience backgrounds; participants with differing SE background could possibly perform differently in the project. To identify any such possible outlier participants, prior to the study, each participant was interviewed about their background experience so that any subjects with prior SE industry experience could be identified. None had any such experience. This, coupled with the knowledge that every participant was a full-time computer science student, and had taken similar software engineering courses for specialization, ensures that they had undergone similar SE training. Furthermore, weekly review sessions were used to identify any obvious outliers during the project, which we did not find.

Differential mortality: This occurs when a physical or mental change occurs to participants during study that is not "equal" between the two types of study groups. This threat existed in our study because of the duration of the participants' project (see Section 3.5), which lasted approximately two months. To contain this threat, the researchers reviewed and assessed weekly submissions of work and collected data. Additionally, weekly motivation meetings were held to further monitor the participants' progress. At the conclusion of the study, all initial participants remained in the study and no effects of the differential mortality threat were observed.

Researcher bias: Occurs when the researcher, knowingly or unknowingly, influences the outcome of the study. This threat exists in our study because of the subjective nature of the requirements characteristics ratings (see Section 3.6.1). To mitigate this threat, multiple researchers and domain experts, and an "open" process (with no direct investment in study), were used in the study processes. These are recognized techniques for dealing with *researcher bias* (Johnson and Christensan, 2003).

3.7.2 External Validity

External validity refers to the degree to which the results of a study can be generalized across a population, time or place (Johnson and Christensan, 2003). Population validity can exist when generalizing to industry; the reason for using students in our study was the *availability sampling* technique. It would have been extremely difficult (if not impossible) to conduct this first-time controlled-study in industry. The use of students should not diminish the results of this study, as important results have been found in other SE studies when student-based studies have been conducted (e.g., in requirements triage (Berander, 2004); code inspection (Carver et al., 2003); and lead-time impact assessment (Host et al., 2000). We do acknowledge the threat in generalizing to experienced requirements engineers; however, there is no evidence suggesting that the results could not be generalizable to, at the very least, novice requirements engineers in industry (Host et al., 2000). Regardless, *exploratory* studies such as this are an important first step towards determining initial results on a particular research issue that can provide the groundwork for future studies in wider contexts.

3.7.3 Construct Validity

Construct validity refers to the extent to which a measurement corresponds to theoretical concepts (i.e., constructs) concerning the phenomenon under study (Johnson and Christensan, 2003). In this study, the constructs were the requirement characteristics. These were measured by an instrument created and used by external researchers (see Section 3.6.1). We held numerous peer-review sessions with a total of six experts to validate the measurement instrument with respect to the theoretical constructs we wanted to investigate (see section 3.1).

3.7.4 Conclusion Validity

Conclusion validity is the degree to which conclusions we make based on our findings are reasonable (Johnson and Christensan, 2003). There are three ways in which conclusion validity can be improved in a quantitative-based study: statistical power, reliability, and proper implementation of study methods. In our study, statistical power (or lack thereof) is not an issue, as our statistical tests are performed on ratings from approx. 900 requirements which were elicited by the 12 RE teams. Also, the study design and statistical tests (see Section 3.3 and Section 4.1 respectively) accounted for the difference between the *unit of analysis* (RE teams) and *unit of observation* (requirements). As discussed in section 3.6.1, we used multiple researchers to rate each requirement in order to achieve a reliability of the rankings. Lastly, the researchers performing the ratings were trained prior to the actual rankings to ensure they properly carried out their task.

4 Data Analysis, Results and Interpretations

We now describe the analysis of the data gathered, the findings and their interpretation. The implications of the findings are described in Section 5.

4.1 Data Analysis

The SA-groups collectively produced 443 newly elicited requirements; whereas, the non-SA groups collectively produced 458 newly elicited requirements. Note that, normally, in a controlled experiment the analysis of data would be conducted on the randomised construct (in our case, the teams). However, we are primarily interested in exploring whether there are significant differences in the "characteristics" of the requirements elicited by the teams (SA vs. non-SA) and not simply in the teams themselves. Thus, the analysis we have conducted acknowledges the team randomisation and "takes into account the extent to which outcomes (i.e., characteristic ratings of the requirements) differ across all the teams "independent of the treatment effect" (i.e., ignoring the presence of SA for the treatment group. This is an established procedure (Rao, 1997)". Specifically, we conducted separate statistical analysis that incorporated the possible effect of the different teams on the characteristic ratings. That is, we performed a 2-way nested ANOVA[25] with the presence of SA as a *fixed* variable and the teams as a *random* variable. This

[25] The 2-way nested ANOVA testing was done using SPSS 16.0 from SPSS Inc. (http://www.spss.com).

statistical analysis corresponds to the experimental design used for this study (see Section 3.3) and is used to test the hypotheses (see Section 3.2). The results of the 2-way nested ANOVA are presented and discussed in the next section. We qualitatively analyse identified architectural aspects that affected the SA-groups' RE process and link them as sources for the differences reported in the preceding analysis.

4.2 Results and Interpretations

We now describe the results and interpretations from the data analysis that was performed. In the following subsection we discuss the results from the hypothesis testing (answer for Q1 – "which requirements characteristics were affected" – See Section 3.1). We then discuss more detailed results regarding each requirement characteristic. Lastly, we present the findings from Q2 (see Section 3.1) where we qualitatively link the SA aspects that were determined to have affected the RE process with the requirement characteristics that showed statistical significance in the preceding analysis.

4.2.1 Requirements Characteristics Hypothesis Testing

Recall that in Section 4.1 we mentioned the 2-way nested ANOVA which tests for the effect, on the requirements characteristics, of both the SA and being in a different requirements team. Here, we now present and interpret the results from this 2-way nested ANOVA. From Table 4-3, we see the characteristics that showed statistically significant difference due to the presence/absence of an SA when controlling for the "team effect" (*focus on user needs, focus on technology needs, architectural relevance* and *importance – all significant at p = 0.000).* This means that there is virtually no possibility that these results were due to chance. Table 4-3 also shows that there was a statistically significant effect from the teams for the characteristics *focus on user needs* and *focus on technology needs* (p = 0.001 and 0.003 respectively). The characteristics *architectural relevance* and *importance* did not show a statistical significant difference for the *team* variable. The characteristics in this table were the only characteristics that showed a significant difference for either the SA or team variable[26].

Requirement Characteristic	Independent Factor	Degrees of Freedom	F-value	Significance
Focus on user needs	SA	2	267.038	0.000
	Team	10	2.914	0.001
Focus on technology needs	SA	2	335.914	0.000
	Team	10	2.688	0.003
Architectural relevance	SA	2	2148.621	0.000
	Team	10	1.233	0.266
Importance	SA	2	3244.578	0.000
	Team	10	1.830	0.052

Table 4-3. Results of nested ANOVA testing

What this means is that both the presence/absence of the SA and being on a different team had an effect on the user and technology-focus characteristics. Because this study was more technically oriented, we did not collect more specific data on why being in a different requirements teams led to an effect on these particular requirements characteristics. However, there are some intuitive explanations for why this interesting phenomenon occurred. It is possible that different individual personal interests and capabilities played a role in changing the "flavour" of the requirements. For example, if the teams had individuals who preferred downstream processes and were thus more "solution" driven (e.g., designing, testing, coding, etc.) then these teams' requirements would have a more technological bias regardless of the presence/absence of an SA. Conversely, if the teams had individuals who preferred the more human aspect of SE (e.g., RE, human-computer interaction, etc.), then their requirements would necessarily be more user-focused. The role of human personality and capability is outside the scope of this study; however, it could prove to be an interesting avenue for future empirical research and is the focus of one of our emerging hypotheses in Section 5.5.

Irrespective of the team effect exhibited above for the two requirements characteristics, we have demonstrated that the SA does have a significant effect on

[26] The ANOVA test can only be conducted on the Likert-based requirements characteristics and not the ordinal-based characteristics (*abstraction* and *type of requirement*). These ordinal-based

the requirement characteristics, and so we discuss the results for the rest of this section at the requirements level which is the focus of this paper.

4.2.2 Detailed Requirements Characteristics Results

We now discuss in more detail the results and their interpretations pertaining to each of the requirements characteristics (described in Table 4-1). Recall, from Table 4-1 in Section 3.1, that 10 of the 12 characteristics (e.g., *focus on cost, focus on time, focus on quality,* etc.) were measurable on the Likert scale; whereas, the remaining 2 (*level of abstraction* and *type of requirement*) were measurable on the ordinal and nominal scales.

	Group	Focus on User Needs	Focus on Tech. Needs	Arch. Relevance	Imp.
Mean	SA	3.26	4.12	4.59	5.63
	Non-SA	3.65	3.42	4.12	5.28
Cohen's Effect Size[27]		Large	Large	Large	Large

Table 4-4. Mean scores of the two types of groups for selected qualities.

4.2.2.1 Technological Needs versus User Needs

On average, the SA-groups scored higher for *focus on technological needs* (4.12 vs. 3.42, Table 4-4); whereas, the non-SA-groups scored higher for *focus on user needs* (3.65 vs. 3.26). There was thus a tradeoff between the characteristics of the requirements from the two types of groups. Usually, when focus on technological needs was high (scoring a 5, 6 or 7 on the Likert scale), focus on user needs was low (scoring a 1, 2 or 3) and vice versa. The notion of this tradeoff is supported by a follow-up test that was conducted, Spearman's rho test, which showed a statistically significant ($p = 0.007$) inverse correlation between the two characteristics. From the

---

characteristics are analyzed separately in Section 4.2.5.

[27] The Cohen's effect size indicates the difference between the two types of groups is "large" (Rao, 1997), meaning that there is not only a statistical difference between the two groups, but that the difference is substantial for real-world application of the results (e.g., making a business decision). What this means in terms of the real-world RE processes and products is discussed in the next subsection.

perspective of RE processes, this data suggests that the SA-groups had *technological needs* higher than *user needs* in 52.9% of their requirements compared to 41.6% for the non-SA-groups. Likewise, the non-SA-groups had *user needs* higher than *technological needs* in 46% of their requirements compared to 37% for the SA-groups. Table 4-5 characterizes the bias towards technological and user needs for each type of group.

The surface-level reasoning for this difference could be that the SA-groups oriented themselves towards the technological arrangement of the system; whereas the non-SA group oriented themselves towards the user perspective of the system. However, the fact that the SA-group, either knowingly or unknowingly, "shortchanged" the user-oriented requirements is rather surprising.

	SA-groups	**Non-SA groups**
Higher focus on user needs	37%	46%
Equal focus	10%	14%
Higher focus on technological needs	53%	42%

Table 4-5. Focus on technological needs vs. user needs

What this means is that the potential benefactors of these requirements (i.e., the various stakeholders) are not having their needs fully met, which could then lead to negative feedback later in downstream processes or when the given product is released, resulting in lower customer satisfaction, poorer product quality, development rework, etc.

4.2.2.2 Architectural Relevance

Another characteristic where the two types of groups scored differently is *architectural relevance* (see Table 4-4), (i.e., the degree to which a requirement will affect the architecture [Bass et al., 2003]). The mean ratings were 4.59: SA-groups vs. 4.12: non-SA-groups. Examining the data more closely, the SA-groups had more requirements (56) [13%] that scored 7 - extremely high than the non-SA-groups (23)[5%]. Conversely, the non-SA-groups had more requirements that scored 1 – 3 (extremely low to slightly low) (163) [37%] than the SA-groups (161) [27%].

The reason for this variance could be that having access to the system's architecture, the SA-groups are better poised to question the architectural-relevance of an elicited requirement in their decision-making. Ultimately, they seem to be have selected more architecturally relevant requirements from their base-set in their solution strategy than have the non-SA groups.

4.2.2.3 Importance

The two types of groups also scored differently with regard to the level of importance of the requirements they produced (see Table 4-4), namely, the degree to which the success of the system depends on the implementation of a requirement. Upon closer examination, the difference for this characteristic is similar to that for *architectural relevance* in that the SA-groups had more requirements that scored 6 or 7 (quite high and extremely high) was 255 (59%) than the non-SA-groups (216) [50%]. The non-SA-groups had more requirements that scored 1 – 3 (extremely low to slightly low) (62)[14%] than the SA-groups (27)[6%]. Both types of groups scored closely for 4 - neither high nor low (42 [10%] for the SA-groups, 46 [11%] for the non-SA-groups) and 5 - slightly high (106 [25%] and 109 [25%]).

This shows that the SA-groups were better able to elicit the kinds of requirements that would be important to the success of the system than the non-SA-groups. This result is surprising because *importance* of a requirement for system success is not influenced only by its technological or user orientation. Other influencing factors include: return on investment, cost, implementability, resource consumption, etc. Rather, these orientations are simply orthogonal. One would thus not expect a statistically significant difference between the SA and non-SA groups. The cause of the difference thus calls for further investigation.

4.2.2.4 Categories with no difference

The means of the ratings for the requirements from the SA-groups and non-SA groups were similar in a number of categories: *focus on time, cost,* and *quality*; *implementability*; and *focus on client needs. Testability* was higher for the SA-groups at p = 0.06, close to a statistically significant result, and so it is a characteristic of interest for future studies. Thus, for these six characteristics, there is no evidence to support the rejection of the null hypothesis (*the presence of a SA has no impact on characteristic* – see Section 3.1).

What this means is that the characteristics of the requirements gathered in the presence or absence of the architecture by the respective two types of groups were statistically not different in so far as these six particular characteristics are concerned.

4.2.2.5 Abstraction and Type

Two of the requirements characteristics (see Table 4-1), *requirement type* and *level of abstraction*, were rated on a nominal and an ordinal scale respectively[28].

For the characteristic *level of abstraction*, the SA-groups had more requirements that scored high values (5 and 6) (71 [16%] vs. 38 [9%]), which denote requirements having a greater breadth of impact (sub-system level, system level, inter-system level). The non-SA-groups had more requirements with lower scores (0 and 1) (58 [13%] vs. 37 [9%]), indicating a small breadth of impact (module or component level). The NHST reveal that the difference in frequency counts is statistically significant ($p = 0.001$) so we conclude that there is evidence to support the rejection of the null hypothesis for this requirements characteristic.

An inference of the SA-groups' higher score on the *level of abstraction* quality over non-SA groups' is that, not having access to the architecture, the non-SA groups were dealing with requirements at a functional level which are dealt with at an individual component's level. On the other hand, the SA-groups' requirements are more cross-cutting across the architecture, since they elicited requirements that dealt with the integration of new sub-systems and components with the existing SA.

For rating the characteristic *requirement type*, requirements were analyzed according to *a priori* nominal categories from standard RE literature (Kotonya and Sommerville, 1998). Examples of these categories include *usability, performance, delivery, reliability, etc.* The dataset shows that the SA-groups produced more implementation (62 vs. 50) and interoperability (51 vs. 30) requirements; whereas, the non-SA-groups produced more usability (51 vs. 20) and functional (99 vs. 81) requirements. NHSTs for this characteristic (*type of requirement*) show that the

[28] Because of the scale types used for these two attributes, the Cohen Effect Size tests cannot be applied.

differences between the two types of groups are statistically significant (p = 0.013), so we conclude that there is evidence to support the rejection of the null hypothesis for this requirement characteristic.

The observation here is consistent with our earlier observation (see Section 4.2.2.1.) that the SA-groups were more focused on back-end technical issues; whereas, the non-SA-groups were more focused on user issues.

4.2.3 SA Aspects vs. Requirement Characteristics

To answer Q2 (see Section 3.1) we can now link aspects of the SA identified by the SA-groups as affecting the specific requirements characteristics that showed a statistically significant difference (see Section 4.2.1). First, requirements of the SA-groups will be divided according to whether or not they were affected by the SA[29]. Thus, we now have three sets of data: (i) those of the SA-groups which *were* affected by SA, (ii) those of the SA-groups which were *not* affected by SA and (iii) the requirements of the non-SA-groups. This division will be used to show that the differences measured between the two types of groups are a direct result of the requirements being affected by the architecture. That is, the presence or absence SA is the "cause" of the specific types of differences observed between the two types of groups.

To confirm that the six requirements characteristics identified in Q1 (see Section 3.1):

- *focus on user needs*, *focus on technological needs*, *architectural relevance*, and *importance*
- *type* and *level of abstraction.*

were indeed affected because of the SA, we now qualitatively examine the differences in the three sets of data identified above (i, ii and iii) with respect to these six requirements characteristics.

Subsection 4.2.3.1 discusses these six requirements characteristics and the differences with respect to them in the three identified groups, and Subsection 4.2.3.2

[29] In total, there were 148 requirements that were *enabled* by the architecture, 126 requirements that were *constrained* and 51 requirements that were *influenced.* However, since we found no statistically significant differences between enabled, constrained and influenced requirements they will simply be

relates specific architectural aspects and their impact on the above six requirement characteristics.

4.2.3.1 Causal Impact of SA on Requirements Characteristics

Earlier findings (see section 4.2.2) indicated that requirements from the SA-groups generally scored lower with regards to *focus on user needs* and higher with regards to *focus on technological needs, architectural relevance* and *importance*. Table 4-7 shows this along with the statistical significance of this new distribution.

On average, when the architecture was not affecting a requirement (188 requirements), both the SA and non-SA groups scored similarly for three characteristics: *focus on user needs*, *architectural relevance,* and *importance*. For these three characteristics, it can now be seen that the differences in means that were observed earlier between the two types of groups (SA groups: 3.26, 4.59, and 5.63 vs. Non-SA groups: 3.65, 4.12, 5.28 respectively -- see Table 4-4) were caused almost entirely by *affected requirements (255* in Table 6). That is, we see "decreased" architectural effect on the characteristic *focus on user needs*, and "increased" effect on the characteristics *architectural relevance,* and *importance.* For *focus on technological needs*, the difference in means reported earlier (SA groups: 4.12 vs. Non-SA groups: 3.42 in Table 4-5) was not caused entirely by *affected requirements* but was certainly augmented by them (SA group not affected: 3.69 vs. SA group affected: 4.4 in **Table 4-6**).

For the characteristics *level of abstraction* and *type,* we did not find any causal link between (i) the differences between the two groups (see Section 4.2.2) and (ii) the SA.

4.2.3.2 Architectural source of the impact

In the previous section, we saw which particular requirements characteristics were affected by the SA (see **Table 4-6**). However, we do not know as yet which particular aspects of the SA were the causes of those effects – this is the focus in this subsection. Determining specific aspects of the SA affecting particular requirements characteristics can help during future elicitation of requirements, for example, in

grouped as *affected requirements.*

being vigilant about any architectural implications on development cost, system quality and schedule and, accordingly, negotiate the requirements with the stakeholders.

Group	Role of SA	Number of Reqt's.	Focus on User Needs	Focus on Tech. Needs	Arch. Rel.	Importance
Non-SA	Not affected	458	3.63	3.42	4.12	5.28
SA	Not affected	188	3.56	3.69	4.24	5.37
	Affected	255	3.1	4.4	4.8	5.8
NHST p-value		Chi-square	0.046	0.001	0.001	0.003

Table 4-6. Characteristics of requirements and the architecture's effect.

In **Table 4-7**, we can see that four specific SA aspects affected requirements that exhibited the properties in **Table 4-6**: *existing hardware*, *non-functional characteristics (same sub-system)*, *non-functional characteristics (different sub-system)* and *architectural patterns*. Requirements affected by one of these four architectural aspects showed lower mean values for *focus on user needs* and higher mean values of *focus on technological needs*, *architectural relevance* and *importance*. This suggests that these four architectural aspects had a substantial and consistent impact on the requirements elicited by the SA-groups.

Modifiability was another SA aspect that affected requirements exhibiting three of the four characteristics (i.e., scored higher means than the groups not affected by SA): *focus on technological needs (4.26)*, *architectural relevance (5.12)* and *importance (5.98)*. However, unlike the four SA aspects described above, these requirements scored *higher* for *focus on user needs* than did requirements which were not affected by the architecture (3.95 vs. 3.56). Without further data and analysis it is difficult to discern the extent of the impact of *modifiability* on *focus on user needs.*

			Requirements Characteristics				
Group	SA Role	SA Aspect	number of reqt's	Focus on User Needs	Focus on Tech. Needs	Arch. Relevance	Importance
Non-SA	None	N/A	458	3.63	3.42	4.12	5.28
SA	None	N/A	188	3.56	3.69	4.24	5.37
	Affected	Existing Hardware	9	3.00	5.67	5.11	5.89
		NF characteristics (same sub-system)	53	1.94	5.32	4.77	5.85
		NF characteristics (different sub-system)	100	3.29	4.55	4.88	5.93
		Arch. Patterns	25	2.88	3.80	4.54	6.25
		Modifiability	42	3.95	4.26	5.12	5.98

Table 4-7. The source of architectural effects and the affected requirements characteristics[30].

4.3 Summary of Results

The following are the key results of the study:

- Given access to the architecture, the analysts tend to elicit approximately 10% more technologically-oriented requirements; without access to the architecture, they tend to elicit approximately 10% more user-needs oriented requirements.
- For a given type of group (SA or non-SA), there is an inverse relationship within the set of elicited requirements between the characteristics technological needs and user needs; as the quantity of one increases the other decreases (e.g., requirements focused on user needs have less focus on technology needs – 46% vs. 37% respectively).
- Given access to the architecture, the analysts tend to elicit 10% more architecturally relevant and10% more important requirements.

[30] The numbers in this table do not equal the totals in Table 6 because Table 6 includes requirements affected by all nine identified architectural aspects, whereas Table 7 contains only those five aspects that had an impact on requirements with differing characteristics.

- Given access to the architecture, the analysts elicited approximately 7% more abstract requirements (i.e., those with cross-cutting concerns across system requirements) than analysts without access to the architecture.
- Given access to the architecture, the analysts elicited more *implementation* (62 vs. 50 requirements) and *interoperability* (51 vs. 30) requirements.
- Without access to the architecture, the analysts elicited more requirements of type *usability* (51 vs. 20) and *functionality* (99 vs. 81).
- Specific architectural aspects were identified that affected the requirements characteristics: *Existing hardware, NF characteristics (same sub-system), NF characteristics (different sub-system), architectural patterns,* and *modifiability.*

Until now, there was no scientific data on the above issues. This can therefore be considered an important step towards building a grounded theory on the impact of SA (or non-SA) on RE. While the findings may be interesting, it is rather unfortunate that the various SA and non-SA groups had long disbanded and therefore not accessible by the time the data analysis was completed. Thus, we couldn't obtain their perspective of our inferences. The next section discusses example implications of our findings.

5 Implications

We discuss the implications of the described findings on such issues as RE process engineering, post-requirements analysis, RE tools, traceability management, and further empirical work in RE.

5.1 RE Process Engineering

The findings described in Section 4.3 raise some interesting questions, such as: should the SA always be used in the RE process as promoted by the literature (Shekaran, 1994; Mead, 1994), and as strongly supported by our survey results (see introduction)? Could there be some conditions when it would be advisable *not* to use SA in RE? Of course, the considerations behind these questions are such factors as project costs, time-to-market, system quality, profitability, innovation, sustainability, and human factors.

We have identified three key cases which merit consideration in the design of RE processes: (1) new innovative system; (2) mature system; and (3) system with user and technology balance.

When evolving a relatively new product, the business strategy might be to focus more on innovative features (so as to attract a large customer base) than on refining technical system attributes such as: reliability, security, performance, etc. (so as to stabilise the system) (Rajlich and Bennett, 2000). In this scenario, management can determine whether the cost/benefit of the RE process would be better if the influence of the existing SA on elicited requirements was *omitted* or *minimized*, if not entirely in the RE process then at least at the outset of the RE process.

Likewise, in a mature product, with a large dependent customer-base, it would be imperative that new requirements do not destabilise system quality, and, accordingly, it would be advisable to use SA in RE. For example, we saw that involving SA in RE led to more global or architecturally-relevant requirements (see Section 4.2.2.2), whereas, absence of SA in RE in this scenario could lead to myopic requirements. In turn, this could lead to increased development backtracking to fix architectural problems or duplication of features (both functional and non-functional) across the system (Kamiya et al., 2002).

In the cases where there is a need for a user *and* technology focused requirements, we need to design the RE process to have mechanisms built-in to ensure that the characteristics of the requirements are not lop-sided in favour of SA and against user-focus (or vice-versa). For example, in section 4.2.2.1, we saw that involving SA in the RE process short-changed user-focused requirements. Consequently, not having vigilance about the impact of SA on RE could result in system qualities that may not satisfy diverse stakeholders' interests.

These are but specific examples. There are many project and organizational factors that could influence when to utilize the SA in a RE process, for example: size and competency of the development team, development paradigm used (e.g., agile vs. iterative), familiarity of SA by the development team, budget, etc. Therefore, in deploying the RE process, these multitude of factors should be considered when planning the inclusion of the SA in the RE process.

5.2 Post-requirements Analysis

As permitted by a project-specific situation, it is prudent to examine the requirements being elicited – as a post-RE or post-project exercise -- to detect any biases counteracting business goals, which could then be adjusted in the subsequent elicitation efforts. For example, by integrating the rating (see Table 4-2 in Section 3.6.1) into post-requirements (or post-development) analysis, a project could gather (release-based or timed) quantitative data on requirements characteristics. This would create a history of the "flavour" of the system, as dictated by the characteristics of the requirements (e.g., trends on quality-attribute biases, user-needs focus, technological focus, etc.). By analysing such trends, management can assess, periodically, whether the evolving system is aligned with the current and future organizational goals. For example, a trend heavily in favour of "user-focus" coupled with heightened architectural defects could indicate inadequate consideration of SA during the RE (and development) processes. Based on such analysis, tweaking the requirements and the associated RE processes can help align the requirements characteristics to the system and business goals. Figure 4-1 shows an example process model of how the requirements rating method could be integrated in a software development process.

Unlike other implications in this paper, this one is rooted in the empirical procedures of the study and not the findings. There is precedence for this idea. For example, in software effort estimation (Johnson et al., 2000), the collection and analysis procedures of software defects, size and effort data has been integrated into a software estimation tool. Likewise, in software process improvement (Cook et al., 1998), the authors propose the use of historical, exploratory research studies to identify process improvement opportunities in industrial projects. Finally, in software maintenance (Porter and Selby, 1990), the analysis of software metrics (such as development effort, defaults, and component changes) in past releases of a software project is used to automatically identify components most likely to cause rework.

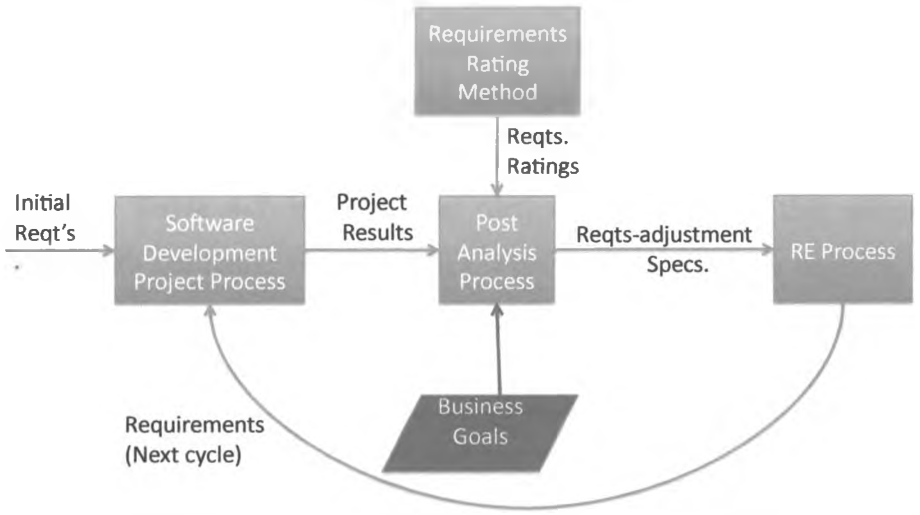


Figure 4-1. Integration of requirements rating method in software development process.

5.3 RE Tools

The realisation that specific characteristics of new requirements are affected by particular aspects of existing system architectures (as depicted in **Table 4-7**) opens up possibilities for a new generation of RE tools. In particular, requirements management tools (such as DOORS and Requisite pro) and goal-oriented modelling tools (such as *i** [Yu, 1997] and KAOS [Lamsweerde, 2003]) could be enhanced with a product-centric knowledge-base that accumulates, over a span of many releases, how different aspects of the evolving system architecture affects requirements characteristics (e.g., as shown for one evolutionary iteration in **Table 4-7**). Such tools thus would have a characteristic of becoming more and more mature over time while enhancing the elicitation, analysis and reasoning capabilities of the requirements engineers.

5.4 Traceability Management

Recent work in (Heindl and Biffl, 2005) encourages value-based requirements traceability, as opposed to early research which attempted at all encompassing traceability (implemented in tools such as DOORS and Requisite Pro) which is known to be wasteful in terms of future use and thus has not gained wide acceptance in practice (Arkley and Riddle, 2005). In this paper, we show how empirical studies can lead to discovery of targeted product dependencies and relationships (e.g., between SA and requirements) that can be worth tracking during software evolution.

In **Table 4-6**, for example, we see that 255 requirements (58%) were affected by the SA. Also, **Table 4-7** indicates the particular SA aspects that affected the requirements with particular characteristics. Furthermore, previous research (Miller et al., 2008) has already shown the different effect types of SA on requirements (e.g., SA as a constraint on new requirements or as an enabler of new requirements). Thus, by collating these different pieces of information, it would be possible to create *targeted* traceable links. For example, by linking those architectural aspects (e.g., components with particular non-functional characteristics) that are historically known to constrain certain types of requirements, it could help in speeding up detection and analysis of *new* risky requirements that are in conflict with the baseline architecture. We can see that empirically derived targeted links would (a) reduce the burden considerably in making the select-few links in the first place and (b) render invaluable information upon usage of traceability tools during RE.

5.5 Further Empirical Studies in RE

Based on the findings of the exploratory study, we raise the following example emergent hypotheses:

H1: Requirements elicited from a RE process that involves analysis of a current architecture will be more technologically focused than a RE process that does not include such analysis.

This hypothesis emerges from the finding in Section 4.2.2.1 and improves upon the null hypothesis used in this study by providing a direction of the effect (i.e., SA analysis implies *more* technological focus). While the finding on technology focus vs. users-need focus may seem intuitive, further testing of this hypothesis in different domains and project contexts would not only confirm whether these results are generalizable across different population and settings (see Section 3.6.2), but would also indicate the variance in *extents* across them. This could help tune the RE process specific to the domain concerned.

H2: Requirements elicited from a RE process that does not analyze the current architecture will be more user-focused than a RE process that does not include such analysis.

In Section 4.2.2.1, the results show that requirements elicited in the absence of an existing architecture are more user-focused than requirements in the presence of an existing architecture. The motivation for this hypothesis follows from H1 above.

H3: Requirements elicited when the current architecture is analyzed are considered more important for system success than without such analysis.

This hypothesis is rooted in the finding from Section 4.2.2.3. This result is surprising and seems more based on factors outside of technology and user needs, and is therefore difficult to discern whether this would hold across other project domains and business contexts.

H4: Requirements elicited when the current architecture is analyzed are more architecturally relevant than requirements without such analysis.

This hypothesis emerges from the finding in Section 4.2.2.4, and the motivation for this hypothesis follows from H1 above.

H5: An RE process that does not include analysis of current architecture will output more innovative requirements.

A requirement characteristic that was not measured in this study was innovation. This was due to the fact that the project domain was banking which is not a new domain, and thus it would be difficult to measure innovation in such a system.

However, it is an important characteristic that we initially did want to investigate. Although this hypothesis is not directly derived from the findings, we include it here because this characteristic should be investigated if the domain permits. By intuition, seems that it could be affected (most likely constrained) by the analysis of a current architecture. The investigation of this hypothesis complements recent research effort (Maiden et al., 2004) in improving the process of eliciting innovative requirements.

H6: A requirements elicitation team with motivation and expertise in system solution is more likely to elicit requirements that have technological bias regardless of the absence or presence of an existing system architecture.

This hypothesis emerges from the finding in Section 4.2.1 where it was shown that the simply being in a different requirements team had a significant effect on the technological bias of the requirements. One possible explanation for this phenomena occurring is that the motivation and expertise of the team members was more solution-oriented and accordingly biased the requirements.

H7: A requirements elicitation team with motivation and expertise in a system's context (e.g., human-computer interaction and end-user satisfaction) is more likely to elicit requirements that user-focused regardless of the absence or presence of an existing system architecture.

This hypothesis emerges from the finding in Section 4.2.1, and the motivation for this hypothesis follows from H6 above.

To test hypotheses H1-H5, controlled experiments with the same type of project setup as described in this paper (see Section 3) would need to be designed. The study design for testing H6-H7 would be more difficult in randomized controlled experiment, and thus, a quasi-experiment design would be more suitable for these cases.

6 Conclusions

While the role of Systems Architecture (SA) in Requirements Engineering (RE) has been discussed several times over the past decade (Shekaran, 1994; Mead, 1994; Jackson, 1994), no scientific studies have ever been conducted to explore the quantitative relationship between SA and requirements. In this paper, we describe a controlled study, involving 12 teams, investigating the characteristics of new requirements in the presence or absence of an existing SA.

In a nutshell, we found that of the 12 requirements characteristics identified (see Table 4-1), the following were significantly affected by the presence or absence of the SA (see Section 4.3): *focus on technological needs*, *architectural relevance*, *importance, level of abstraction, requirement type,* and *focus on user needs*. We did not find SA effects on the remaining characteristics (see section 4.2.2.4). We then probed the results further from the perspective of the specific architectural aspects that affected these characteristics and found five such aspects (*Existing hardware, NF characteristics (same sub-system), NF characteristics (different sub-system), architectural patterns,* and *modifiability).* From these findings, we discuss potentially useful implications in the areas of RE process engineering, requirements alignment with business goals, RE tools, traceability management, and future empirical work based on four emergent hypotheses from this study.

While these findings contribute new scientific knowledge concerning SA-requirements interaction, let us not forget that this was only one exploratory controlled study on a particular domain (Banking system). Significant as it is, we advise caution when making business or project decisions based on the findings of this foundational study alone! Rather, we encourage other researchers to conduct confirmatory studies in other domains and contexts to help build grounded theory on the impact of SA on RE.

Though our study, in principle, can be replicated in both industry and academia, each of these contexts has its own limitations to consider when planning replications of the study. Conducting controlled studies in industry would be extremely difficult, if not impossible, due to the unavailability of equivalent projects, and the inability to impose research control (i.e., presence or absence of SA in the RE process). More likely, the chances are better to conduct controlled studies and perform hypothesis testing in academia using qualified students as participants. This, however, does have the issue of being able to generalize the results to industrial contexts. Despite this threat, studies such as these are still a critical stepping-stone to conducting "case studies" in industry. Case studies in industry would be invaluable for providing an industrial perspective on the impact of SA on RE. These studies can be carried out by analyzing the existing requirements of projects where the RE processes have either involved SA or not. In particular, data from different projects with varying levels of complexity, and in different domains, would help solidify the body of knowledge in this research area.

References

Arkley, P., Riddle, S., 2005. Overcoming the Traceability Benefit Problem. 13th IEEE International Conference on Requirements Engineering (RE'05), Paris, France, pp. 385–389.

Basili, Victor R. and Weiss, D., 1984. A Methodology for Collecting Valid Software Engineering Data. *IEEE Trans.OnSoft.Engineering*, pp 728-738.

Bass, L.; Clements, P.; and Kazman, R., 2003. *Software Architecture in Practice, 2nd edition*. Addison-Wesley.

Bayer, J., Muthig, D., Widen, T., 2000. Customizable Domain Analysis. Proceedings of the First International Symposium of Generative and Component-Based Software Engineering (GCSE '99), Lecture Notes in Computer Science. Springer-Verlag, pp. 178-194.

Berander, P., 2004. Using Students as Subjects in Requirements Prioritization. Proceedings of the 7th International Conference on Empirical Assessment & Evaluation in Software Engineering, pp. 95-102.

Breitman, K., Sampaio do Prado, J. C., 2000. Scenario Evolution: A Closer View on Relationships. Fourth International Conference on Requirements Engineering (RE'00), Illinois, US, pp.95-107.

Burgess, C., Dattani, I., Hughes, G., May, J., Rees, K., 2001. Using Influence Diagrams to Aid the Management of Software Change. Journal of Requirements Engineering, Volume 6, Number 3, pp. 173-182.

Carver, J.; Shull, F.; Basili, V., 2003. Observational Studies to Accelerate Process Experience in Classroom Studies: An Evaluation. Proceedings of the 2003

International Symposium on Empirical Software Engineering (ISESE '03), Rome, Italy, pp. 72-79.

Castillo, E., Gutiérrez, M., Hadi, A., 1997. Learning Bayesian Networks. *Expert Systems and Probabilistic Network Models*. Monographs in computer science. New York: Springer-Verlag. pp. 481-528.

Cook, J., Votta, L. G., Wolf, A. L., 1998. Cost-Effective Analysis of In-Place Software Processes. IEEE Transactions on Software Engineering, Volume 24, Issue 8 (August 1998), pp. 650 - 663.

Egyed, A., Grunbacher, P., Medvidovic, N., 2001. Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach. First International Workshop from Software Requirements to Architectures (STRAW '01), Toronto, Canada, June, 2001.

El Emam, K. and Madhavji, N., 1995. Measuring the Success of Requirements Engineering Processes. Proceedings of the 2nd IEEE International Symposium on Requirements Engineering, pp. 204-211.

Etien, A., and Salinesi, C., 2005. Managing Requirements in a Co-evolution Context. 13th IEEE International Requirements Engineering Conference (RE'05), pp.125-134, Paris, France.

Ferrari, R., and Madhavji, N. H., 2008. Software architecting without requirements knowledge and experience: What are the repercussions?. *Journal of Systems and Software*, Vol. 81, No. 9. (September 2008), pp. 1470-1490.

Ferrari, R., Miller, J., Madhavji, N. H., 2010. A controlled experiment to assess the impact of system architectures on new system requirements. Requirements Engineering Journal, accepted, publication to appear, DOI: 10.1007/s00766-010-0099-3.

Ferreira, S., Collofello, J., Shunk, D., Mackulak, G., 2009. Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation. Journal of Systems and Software, Volume 82, Issue 10, October 2009, Pages 1568-1577.

Fusaro, P.; El Emam, K.; Smith, B., 1997. Evaluating the interrater agreement of process capability ratings. Proceedings of the 4th International Software Metrics Symposium, pp. 2-11.

Heindl, M., and Biffl, S., 2005. A case study on value-based requirements tracing. Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 60-69, Lisbon, Portugal.

IEEE SWEBOK, 2004. Guide to the Software Engineering Body of Knowledge: 2004 Version. IEEE and IEEE Computer Society project. <http://www.swebok.org/>

M. Host, B. Regnell, C. Wohlin, 2000. Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering, pp. 201–214.

Jackson, M., 1994. The Role of Architecture in Requirements Engineering. Proceedings of the 1st International Conference on Requirements Engineering (RE '94'), pp. 241.

John, I., Muthig, D., Sody, P., Tolzmann, E., 2002. Efficient and Systematic Software Evolution Through Domain Analysis. 10th IEEE Joint International Requirements Engineering Conference (RE '02), pp. 237-245, Essen, Germany.

Johnson, P. M., Moore, C. A., Dane, J.A., and Brewer, R. S., 2000. Empirically guided software effort guesstimation. *IEEE Software*, 17(6), December 2000.

Johnson, R. B., and Christensan, L., 2003. Educational Research: Quantitative, Qualitative and Mixed Approaches. www.southalabama.edu/coe/bset/johnson/dr_johnson/2lectures.htm. Date last accessed June 2009.

Kamiya, T., Kusumoto, S., Inoue, K., 2002. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering, July 2002, Vol. 28, Issue 7, pp. 654-670.

Kotonya, G.; Sommerville, I., 1998. *Requirements Engineering*. England: John Wiley & Sons Ltd.

Lamsweerde, A. Van, 2003. *From System Goals to Software Architecture.* In Formal Methods for Software Architectures, M. Bernardo & P. Inverardi (eds), LNCS 2804, Springer-Verlag, pp. 25-43.

Maiden, N., Manning, S., Robertson, S., Greenwood, J., 2004. Integrating creativity workshops into structured requirements processes. Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques, Cambridge, MA, US, pp. 113-122.

Mead, N., 1994. The Role of Software Architecture in Requirements Engineering. Proceedings of the 1st International Conference on Requirements Engineering, pp. 242.

Miller, J., Ferrari, R., Madhavji, N. H., 2008. *Architectural Effects on Requirements Decisions: An Exploratory Study.* 7th Working International Conference on Software Architecture, Vancouver, pp. 231-240.

Miller, J., Ferrari, R., Madhavji, N. H., 2009. Characteristics of New Requirements in the Presence or Absence of an Existing System Architecture. 17th International Conference on Requirements Engineering (RE '09), Atlanta, United States, pp. 5-14.

Nuseibeh, B., 2001. Weaving Together Requirements and Architectures. IEEE Computer, 34(3): 115-117.

Nuseibeh, B., Easterbrook, S., 2000. *Requirements engineering: a roadmap.* Proceedings of the 22nd International Conference on Software Engineering (ICSE), pp. 27-46.

Pett, M. A., 1997. *Nonparametric Statistics for Health Care Research: Statistics for Small Samples and Unusual Distributions*, Edition: 2. Published by SAGE, 1997.

Porter, A. A., Selby, R. W., 1990. Empirically Guided Software Development Using Metric-Based Classification Trees. IEEE Software, Volume 7, Issue 2, March 1990, pp. 46-54.

Rajlich, V. T., Bennett, K. H., 2000. A staged model for the software life cycle. IEEE Computer, Volume 33, Issue 7, Jul 2000 Page(s): 66 – 71.

Rao, P.V., 1997. *Statistical Research Methods in the Life Sciences*, Brooks/Cole.

Rolland, C., Salinesi, C., Etien, A., 2004. Eliciting gaps in requirements change. Journal of Requirements Engineering, Volume 9, Number 1, 2004, pp. 1-15.

Shaw, M., 2003. Writing good software engineering research papers: minitutorial. In: Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), Portland, USA, Tutorial Session, pp. 726–736.

Shekaran, C., 1994. Panel Overview: The Role of Software Architecture in Requirements Engineering. Proceedings of 1st International Conference on Requirements Engineering, April 1994, pp. 239.

Software Requirements to Architectures Workshop (STRAW'01 & '03), June 2001, Toronto, Canada; May 2003, Portland, USA.

Tichy, W.F., Lukowicz, Prechelt, L., Ernst, A., 1995. Experimental evaluation in computer science: a quantitative study. Journal of Systems and Software (January), 1–18.

Vilella, K., Doerr, J., Gross, A., 2008. Proactively Managing the Evolution of Embedded System Requirements. 16th International Conference on Requirements Engineering (RE '08), pp.13-22, Delhi, India.

Wieringa, R.J., Heerkens, J., 2006. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. Requirements Engineering Journal 11, 295–307.

Yu, E., 1997. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97) Jan. 6-8, 1997, pp. 226-235, Washington D.C., USA.

Chapter 5

An Exploratory Study of Architectural Effects on Requirements Decisions[31]

1 Introduction

No one would deny that if we were to extend an existing edifice, many of its functional and non-functional features would be of central importance in considering new requirements for the extension. Yet, in the software engineering (SE) literature, this is rather an understated issue – that is, consideration of existing system design is not a key factor in engineering new requirements. While in software practice many developers are indeed aware of the need to assess the fitness of new requirements with the existing system design, the approaches are rather subjective and experiential. SWEBOK (IEEE SWEBOK, 2004) – the SE body of knowledge – for example, does not describe any practices to deal with this issue. To explore this issue further, we conducted a preliminary survey of 17 professional requirements engineers and software architects. We found that the average rating of the importance of considering existing systems architecture (SA[32]) when engineering new requirements was 4.5 (on a 1-5 Likert-scale) – implying that the respondents strongly agreed with this concept. Despite this, several respondents noted in the qualitative part of the survey that in actual practice, many organizations neglect this consideration, or only perform analysis on existing high-level feature descriptions of the current system, and not the system's architecture. In many situations, a lack of consideration for an existing system in the new requirements work can lead to rework of requirements and design, incurring extensive costs especially if further downstream in the development process (Boehm and Basili, 2001).

[31] A version of this chapter has been accepted for publication in Journal of Systems and Software.
[32] For the rest of the paper, the acronym SA refers to Systems (or Software) Architecture as a software artefact.

The uptake of this, architecture-requirements, issue in research is not impressive either. It was not until 1994 that the role of an existing SA in requirements engineering (RE) was recognised as important in a panel session. However, at that time, "we still [did] not have a clear understanding of [it]." (Shekaran, 1994a). Shortly thereafter, 5 of the 34 identified indicators of RE success were found to have links with SA (El Emam and Madhavji, 1995). A few years later, the question of an architecture's role in RE was raised again (Nuseibeh and Easterbrook, 2000). While the awareness of an architecture's role in the RE process has no doubt increased, to our knowledge, the effects of an existing SA on RE decisions have not been scientifically explored. It is not until such studies are conducted, and a dependable body of knowledge created, that practice can begin to use such knowledge in day-to-day projects. As a first step in this direction, this paper describes an exploratory case study on the effects of an existing SA on RE decisions. Specifically, we ask:

"In which manner does an architecture affect requirements decision-making[33]?".

We explore this question on two fronts: (1) the kind of role a SA plays in requirements decision-making and (2) the specific aspects of the architecture that affect RE decisions.

For point (1) above, it has already been suggested that a SA might *constrain* a RE process (Shekaran, 1994b). For example, while analysts could be eliciting requirements to employ a new technology that requires a specific communication protocol, the current legacy system has long implemented a conflicting communications protocol, thereby *constraining* the current RE strategy. For point (2) above, while SA aspects are likely largely unique to the domain of these cases, they would give us an indication of which parts of an existing architecture can affect RE decision-making (e.g., non-functional SA areas outside the focus of an RE agent) and, consequently, which parts of the architecture are critical to document for use by requirements engineers

[33] Decision-making leads from recognition of a problem to be solved to a specification of that problem or a solution strategy.

Our results indicate that the relationship between SA and RE is more complex than what is intuitively known in the literature. In particular, "SA as a constraint" is only one of the four types of effects observed in our study. The other three types of effects we found are: *enabled*, *influenced* and *neutral*. In short, an *enabled* effect is where the proposed solution (denoted by the new requirements) is made feasible because of the implemented decisions in the existing system; an *influenced* effect is where the architectural configuration has an effect on the requirements decision without affecting its feasibility; and a *neutral* effect is where there is no noticeable architectural effect on the decision. This paper gives quantitative measures on these effects from the study and qualitative interpretation of the findings. Also, in our study, nine architectural aspects were identified across 117-recorded decisions. Again, this paper gives quantitative measures and qualitative interpretations.

A deeper understanding of the role of SA in RE could open up new opportunities for RE and architecting methods, tools and processes. For instance, in the area of planning and risk assessment, the management could make more informed cost estimates of new requirements by considering how the SA has historically affected the various types of requirements. Likewise, in the area of technology improvement, RE and SA tools can be integrated so that analysts and architects can share, access and change requirements and architecture information more easily. We describe several other cases in the paper.

Our empirical study involved six RE teams that gathered new requirements for an existing system and were observed over the course of two months. The project was in the banking domain and required the RE teams to elicit and analyse new requirements based on a set of high-level features that needed to be integrated into the current architecture. A requirements-decision meta-model was created as a basis for the development of a requirements-tool that served to gather data from the participants during the project on how requirements decisions they were making were affected by specific aspects of the existing architecture. This paper describes: the study context, participant details, project work involved, the underlying decision meta-model[34] for the data that is gathered, use of tools for gathering data, and the various threats to validity.

The key results are the quantitative characterization of the different interaction effects mentioned earlier. For example, for this particular system, nine SA aspects affected approximately 60% of the RE-decisions. From the findings, we have derived four hypotheses that provide a basis for future studies. A general description of how each of these studies could be conducted is also described.

This paper is structured as follows: Section 2 discusses related work; Section 3 describes the exploratory study; Section 4 presents the results; Section 5 discusses various implications from the results; Section 6 discusses future empirical work and emergent hypotheses from this study, and Section 7 concludes the paper.

2 Related work

This section describes the work that is related to our study. The section focuses on three key aspects: (i) observations, commentary and empirical work on the relationship between RE and SA, (ii) technological research spanning RE and SA, and (iii) recent technological-based research on architecture evolution. In subsection 2.4, the section concludes with a reflection on the current state of research described in subsections 2.1 to 2.3.

2.1 RE and SA relationship

There is an increasing interest in exploring and refining the transitions between various activities in the software development process. In particular, the relationship between RE and SA, and their impact on each other was the focus of a couple of workshops five-to-seven years ago (STRAW, 2001; STRAW, 2003). In fact, even earlier, Jackson argued in a panel session (Jackson, 1994) for a tight coupling of the RE and SA processes, suggesting that the most successful developers are those who are able to move relatively more freely between stages within the development cycle. In (Kozaczynski, 2002), Kozaczynski discusses that a level of foresight on the part of

[34] The decision meta-model defines the type of data relevant to this study and is a basis for the tool developed for data gathering. The meta-model and tool are bi-products of this study.

architects to focus on those requirements that are architecturally relevant can help to mitigate development risk in the software process, by being able to develop the architecture early without all requirements being elicited. This, early development, can then be fed-back to the requirements process to further refine the requirements.

In our earlier work in (El Emam and Madhavji, 1995), El Emam and Madhavji presented an instrument for measuring RE success. Through an industry field study to design this instrument, we found that in evolutionary work, the level of understanding of the existing architecture can have an impact on the success of the RE process. In understanding the architecture, requirements engineers can provide requirements solutions that are consistent with the current technical and corporate orientation of its organization. In turn, this can lead to better cost/benefit analysis during RE. This early understanding, however, did not delve into the type of technical effects an existing architecture has on RE decision-making; in this paper, we investigate this issue further.

In (Garlan, 1994), Garlan recognizes that architectural families constrain system requirements. Further, he identifies that solutions can drive requirements. For example, the architecture of a family of systems determines the range of variability allowed in a product line. Though not explicitly stated, one can interpret this as not only architectures imposing “constrains” on requirements decision-making, but also as “enabling” and “influencing” such decision-making. This is a central aspect of the current paper.

In (Bass et al., 2003), Bass et al. discuss that different stakeholders of the architecture will have different needs for documentation, and the level of detail provided to them should reflect this. Depending on the stakeholders’ needs, they can be provided with *detailed information*, *some details* or *overview information* of the various architectural views available. The specific architectural aspects that could be important in RE, however, are not mentioned in (Bass et al., 2003); our study uncovers these details.

Three previous studies of ours, described below, empirically examine RE and SA interaction issues from the viewpoints of: architecting problems rooted in requirements, the effect of using different types of human agents when architecting, and the impact of an SA on requirements characteristics. In (Ferrari and Madhavji, 2008a), we report on a multiple-case study that investigated requirements-oriented problems that are encountered while architecting. Overall, we found that approximately 35% of the problems encountered during architecting were requirements-oriented. Also, specific problem areas together with their severity were identified (such as, quality satisfaction, requirements understanding and quality drivers) as well as the relative frequency of problems occurring in these areas. Implications of this work are on improving methods, tools, and techniques to transition from requirements to architecture. In another study, described in (Ferrari and Madhavji, 2008b), we report on a controlled-study that investigates the impact of software architects having RE knowledge and experience when performing SA. Specifically, two types of study groups were used, the one type of group had previous training and/or experience in RE, and the other type of group did not. Both types of groups conducted the same architecting project given the same initial set of requirements from the banking domain. The results show that the architects with RE knowledge/experience produced a significantly better architecture (10% difference in the overall architectural quality), and the study also highlighted specific architecting areas where these architects performed better. Examples of these areas include: determining architectural tactics, selecting/creating an architectural pattern to satisfy key quality drivers, and interface specification. In a more recent study of ours (Miller et al., 2009), we report on a controlled-study that investigates the impact an SA has on the characteristics of newly elicited requirements. Two types of study groups were used and conducted the same requirements project. One type of group had access to a previous SA whereas the other type of group did not. The results showed that_a multitude of characteristics (e.g., end-user focus, technological focus, abstraction, and importance) were significantly affected by the presence or absence of an SA, and the results also showed extent of this effect. Implications of this work are on RE process engineering in the contexts of new development and legacy systems, and on post-requirements analysis.

2.2 RE and SA technology

There is a growing body of technological work (e.g., methods, software tools, processes, development paradigms, notations, etc.) that is aimed at bridging the areas of RE and SA (STRAW 2001; STRAW 2003). The study presented in this paper is meant to elicit new findings regarding the RE and SA interplay that could then possibly be used in improving such technologies.

Bass et al.'s stakeholder-centred Attribute-driven Design (ADD) method (Bass et al., 2003) focuses on iteratively building architectures based on the key architectural drivers of the system. These drivers are composed of key requirements and quality scenarios that shape the architecture. The drivers are input into the process where architectural patterns are created/selected to realize the tactics (i.e., the architectural design choices made) which, in turn, are aimed at satisfying the quality scenarios. Tradeoffs emerge in the patterns between various quality attributes, and the architects and other stakeholders must negotiate a resolution to these tradeoffs (similar in principle to the Architecture Tradeoff Analysis Method (ATAM) (Kazman et al., 2000) to finalize patterns that would represent an architecture that is most suited to meet the system's goals. Recently, a prototype tool, called ArchE (Diaz-Pace et al., 2008) has been developed to provide support to the ADD method. This support is in the form of modelling the functional responsibilities of the architecture, storing the quality scenarios, and through analysis of the architecture and quality scenarios, the tool suggests tactics that can be used to satisfy the quality requirements. To date, the tool supports modifiability and performance quality attributes, but provides plug-in support so users can add reasoning and analysis frameworks for other quality attributes.

In our previous work, we had developed a method that traces architectural concerns back to the requirements -- the Architecture-centric Concern Analysis Method (ACCA) (Wang et al., 2005). The method uses a Concern Traceability map (CT-map) that captures and presents architectural design decisions starting from software requirements through to the systems architecture and these are then linked to architectural concerns that are identified in the architecture evaluation phase. Through a visual, decision-based, model this method aids in identifying potentially problematic, or sensitive, requirements or decisions that resulted in the concerned architectural parts.

Egyed et al., in their CBSP (Component-Bus-System and Properties) method (Egyed et al., 2001), use an intermediate language (and tool-support) for expressing requirements in a form that more closely relates to architecture, where requirements are identified and categorized based on various properties such as whether they should be implemented as components, bus, system properties, and so on. This method is focused on early architecting work and is not intended for the entire architecting process. In (Hofmeister et al., 2005), the authors deal with the identification and analysis of global factors - those that take into account more holistic issues such as the environment in which the system is built, developing organization, external technological solutions, flexibility or rigidity of requirements, and more. Their two-phase method is a means to design and describe a high-level architecture, and analyse and resolve architectural issues introduced by global factors. In particular, the second phase of their approach (Global Analysis phase), explicitly captures alternative high-level architectural strategies with decomposed design decisions and supporting rationale, and also provides traceability to the requirements.

In (Bruin and Van Vliet, 2003), the authors propose an architectural design method called Quality-Driven Architecture Composition (QAC) where the emphasis is on the reuse of architectural solutions. Their method is iterative and starts with the design of an architecture -- based only on functional features – and where variability points of the architecture are identified. These variability points are expected to cater to the non-functional requirements. The authors call this initial design the "reference architecture". Next, the method focuses on the non-functional requirements by iteratively applying known design solutions (i.e., architectural and design patterns). The Feature-Set (FS) graph (which contains pre-existing knowledge about the domain -- expressed as requirements) and the resultant design fragments (with their accompanying rationale, assumptions, etc.) that can satisfy the requirements drive this entire process. In (Farenhorst et al., 2007), the authors report on a case study that was conducted to explore practitioner's needs for tool support that focuses on architectural knowledge. The study found that practitioner's require a tool that provides "just-in-time" architectural knowledge, defined as access and delivery of the pertinent architectural knowledge to the right person at any given point in time. Given this broad requirement, the authors developed an architectural knowledge-sharing portal that stores various types of architectural knowledge and allows for near-instant retrieval through integrated codification techniques.

In (Stoll et al., 2008), the authors present the Influencing Factors method that guides architects in transitioning from high-level stakeholder concerns to preliminary architectural decisions. An "Influencing Factor" is any stakeholder concern that is considered to play an influential role on the architecture. These influencing factors can be derived from, to name a few, software quality attributes, business goals, market trends, project experience, etc. The method itself has three main steps: identification of influencing factors, which is accomplished through interviews and workshops with stakeholders; prioritization of the influencing factors; and lastly, the factors are analysed with respect to their impact on software quality attributes, which can then aid the architect to make preliminary architectural design decisions.

Cui, et al. (Cui et al., 2008) present an architectural design approach that is also aimed at transitioning from requirements to architecture through the automatic synthesis of candidate architectural solutions. The authors construct their approach on a meta-model that models issues (architecturally relevant requirements), architectural solutions, rationale, and architectural decisions and their relationships. The authors argue that these elements are the key notions for architecture design and the derivation of target architectures. The approach itself has four phases. In the first phase the system stakeholders elicit all possible issues (i.e., architecturally relevant requirements). In the second phase, the architects derive candidate architectures for each issue. The third phase involves the use of a formal grammar that facilitates the automatic synthesis of the candidate architectures developed in the previous phase. These architectural solutions are then presented to the architects in the final phase who can then decide to adopt or reject various aspects (or the entire architectures) and provide rationale for their decision which is then stored for future architectural development iterations.

In (Schwanke 2005), Schwanke discusses the "Good Enough Architectural Requirements Process" (GEAR). This process is meant to further refine an initial set of requirements through architectural means. The process is based on three architectural requirements engineering approaches: model-driven requirements engineering (where elicited candidate-requirements are modelled as use cases, activity diagrams, state charts, etc.), quality attribute scenarios (used to elicit, document and prioritize stakeholder concerns), and global analysis (a general way of organizing information about the problem context that surrounds the architecture). The main purpose of the process is to show where the above approaches overlap and where they complement each other, providing insight into the identification of architectural requirements.

Rapanotti et al. (Rapanotti et al., 2004) propose the extension of "problem-frames" into "architecture frames", which capture information about architectural styles and their interaction with the problem space. The benefit of this mechanism is that in introducing solution-oriented approaches early in development, one can refine problem analysis.

2.3 Architecture evolution

An area of research that is related to our work is architecture evolution, in particular from the viewpoint of methods, processes, and tools development. In the following subsection we highlight recent research in this area; later in Section 5, we discuss how our study can benefit architectural evolution research.

In (Keuler et al., 2008), the authors propose an approach for performing quality impact analysis on an SA. Their approach uses an aspect-oriented solution to automate integration of automated integration of specific concerns (e.g., performance) into architectural models, providing specific quality impact evaluations. This approach is structured in four phases, the first two which can be executed concurrently: (1) architectural styles are applied to create an initial style that is specific to the product architecture; and (2) quality models for the key quality drivers are created, along with their accompanying evaluation models. In the third phase, aspects are used to automatically connect the quality models to the existing architecture; and, in the fourth phase, quality specific views are extracted from the integrated architectural models and are assessed against the evaluation models from (2). The output of this approach is an identification of the specific parts of the architecture that are affecting the achievement of quality attributes. The architect can then use this information for planning changes to the architecture as appropriate.

In (LaMantia et al., 2008), the authors provide case study results from two architecture evolution projects examined over multiple releases where, in each project, architectural modeling was aided by design structure matrices and in accordance with Baldwin and Clark's *design rule theory* (Baldwin and Clark, 2000). Design rule theory is a formal theory that explains how *design rules* (such as splitting or substituting modules) can be used to resolve interdependencies and create modular architectures by specifying the interface between modules. Design structure matrices were designed to support this theory, and are a means of formally modeling interactions between modules of engineered systems. In short, design structure matrix is a square matrix, in which each module corresponds both to a row and a column of the matrix. A cell is checked if and only if the design decision corresponding to its row depends on the design decision corresponding to the column. Based on the two case studies results, the authors argue that the use of design structure matrices and design rule theory improved the modifiability of the systems by (1) allowing for different concurrent levels of evolution in different modules with no negative consequence on system or development process, and, (2) facilitated the substitution of risky components with newly proposed components without substantial change to other parts of the system. The authors conclude that the functionality of design rule theory and design structure can be expanded to provide prescriptive and predictive power in software evolution. Specifically, that the technology could be used to proactively plan for system refactoring.

In (Shen and Madhavji, 2006), the authors propose a method for developing evolutionary scenarios that provide information concerning the impact different types of historical changes (e.g., those related to specific functionality, or those related to external concerns such as security, performance, availability, or those due to internal concerns such as maintainability, system defects, etc.) have had on the quality of software architectural elements of interest. Software maintainers, in particular software architects, can use this information when planning system changes. For example, if the maintainer receives a request for a performance modification, they can consult the scenarios to determine the past impact of performance modifications on the modifiability of the system. The scenarios could suggest, for example, that a major refactoring job was required for most past performance modifications, so the maintainer can then plan accordingly the resource and time allocation to complete the performance modification and any accompanying changes. The evolutionary scenarios focus on the different types of changes that have historically affected a given architectural element at different times in the evolution of the system. This affect is indicated by measures of the quality of the component, for example performance, fault-proneness, level of maintainability, etc. The scenarios also provide information on which component sets that have been affected by a given type of change at different times in the evolution of the system. To create these evolutionary scenarios, the *Evolutionary Scenario Development Method* was designed. This structured and automated (where possible) method is needed since the data sources on which the scenarios are constructed can be quite large. The possible inputs to the method can include: bug reports, CVS data, source code, change log fixes, architectural design documents and feature requests. Currently, the method and supporting technology facilitate building evolutionary scenarios that have a focus on maintainability and fault-proneness.

The above work describes research that is focused on performing "off-line" evolution, which basically assumes that the system can be shutdown to perform and integrate the new changes. Other recent research in the area of architecture evolution proposes technology for performing automated run-time architectural evolution.

In (Wagnier et al., 2007), the authors propose a framework for performing automated architectural evolution. Specifically, they detail FIESTA, a framework that aids architects in adding new functionality when performing architectural evolution. Their framework is generic in that it allows an architecture to be specified in any architectural description language. The framework functions by taking as input a formal specification of the new functionality to be added and the architect then decides where in the existing architecture the new functionality should be integrated. The system then automatically makes the transformation into a modified architecture. In another automated architectural evolution approach, described in (Morrison et al., 2007), the authors propose a formal architectural description language called Archware-ADL that facilitates *active architectures*, namely an architecture that can be evolved automatically during system run-time based on both internal system and external changes. The basic premise of the language is to formally model the architecture as part of the on-going computation, thereby allowing evolution during execution. Developers can express new components, connectors, constraints and evolutionary rules in this notation and initiate integration with the system. The system will then accordingly modify and monitor system, without any downtime. The authors also propose a set of support technologies to support this language for these evolutionary purposes.

2.4 Reflection on research

The previous three subsections discuss research in the primary areas that are related to our study: current knowledge pertaining to the relationship between RE and SA; technology aimed at transitioning from RE to SA; and, architecture evolution. In this subsection, we reflect on the current state of research in these areas.

As discussed In Section 2.1, as early as 1994, researchers discussed the importance of the role of an SA in RE (Shekeran, 1994). A few other works have commented on this issue since then (El Emam and Madhavji, 1995; Nuseibeh and Easterbrook, 2000), and also other knowledge-seeking empirical studies have been conducted in the area of the RE and SA relationship (Ferrari and Madhavji, 2008a; Ferrari and Madhavji, 2008b; Miller et al., 2009). However, beyond these works there has been, to our knowledge, sparse research conducted in the area of the role of an SA in RE. When looking at the other direction in the RE and SA relationship, i.e., transitioning from RE to SA, there is an abundance of research work conducted in this area, particularly with a focus on technological approaches. In the RE and SA interaction technological works described in Section 2.2, there is an implicit assumption that the development is starting from “scratch” i.e., there is no existing system that is being enhanced. In industrial practices, however, software development is largely conducted within evolutionary processes, (IEEE SWEBOK, 2004). Conversely, the research work presented in Section 2.3 (Architecture Evolution) is focused on the improvement of the architecting process in the context of an evolving system. However, this work solely focused on architecting; the RE process is not explicitly considered during architectural evolution and is treated as a "black-box" process where requirements are simply input into the architecture evolutionary processes. Therefore, there is little to no consideration in this research for the RE -SA interaction as highlighted in the works from Sections 2.1 and 2.2.

Thus, there is a need to consider the current system explicitly when performing RE and SA. Furthermore, there is a lack of empirical evidence regarding the specific interaction effects between RE and SA. The empirical study presented in this paper is meant to present detailed quantitative findings on the effect of the presence of a current architecture when performing RE. Such findings can be fed back into research on state-of-the-art technologies (such as the work described in Sections 2.2 to 2.3) to facilitate improvement in RE and SA evolutionary processes.

Though the importance of conducting empirical studies in software engineering (SE) has been recognised (Tichy et al., 1995; Wieringa and Heerkens, 2006), Shaw's analysis (Shaw, 2003) of research papers submitted at a prominent 2002 SE conference suggests that only 12% were submitted in the category of "Design, evaluation, or analysis of a particular instance" and 0% in the category of "Feasibility study or exploration". In (Ferrari and Madhavji, 2008b), we presented our own analysis of published papers. In the fields of RE and SA, since the year 2000, only approximately 15% of the published papers were in the above-mentioned categories, suggesting that studies such as the work described in this paper are currently rather rare. Our work is meant to help in filling this research gap.

3 The study

Exploratory studies are used when the "research looks for patterns, ideas, or hypotheses rather than research that tries to test or confirm hypotheses" (Vogt, 1993). The current research about architectural effects on RE decisions has been anecdotal (Nuseibeh, 2001; Shekaran, 1994a), and thus there is not much grounded theory on this subject. Our study fits the exploratory study characteristics. By having multiple cases, we are able to identify trends and patterns beyond a single-case study design.

The following sub-sections deal with: the research questions, participants, the requirements project, data collection, and threats to validity.

3.1 Research questions

Recall from the Introduction section that the intent of this case study was to investigate the role of an architecture in requirements decision-making. We thus have two pertinent research questions:

Q1: *How does an architecture affect requirements decision-making?*

This question deals with the impact the presence of an architecture has on decision-making in RE. This is accomplished by asking the participants of this study, for every decision that they make, how has the architecture affected that decision. By having a quantitative profile of various architectural effect types, we can investigate improvement to RE and software architecting technology with the help of this new information.

Q2: *Which aspects of the architecture affect requirements decisions?*

This second question is intended to probe into the details of Q1. Whereas Q1 was aimed more generally at the effect of architecture on requirements decisions, this question aims to characterize the various architectural aspects that are found to have an effect. Through characterization of the different architectural aspects, we can begin to examine improvement opportunities during architecting that can optimize future requirements work on a system.

A purposeful tool was developed to gather the data for both the research questions Q1 and Q2 above. The tool is discussed in Section 3.4.2.

3.2 Participants

The population of this study is requirements engineers working in the evolutionary phase (i.e., after the initial release) of a system. The participants of the study were 12 graduate and final-year undergraduate level computer science students at the University of Western Ontario who were randomly assigned to 6 teams, each composed of 2 members. The external validity threat from using students in studies is discussed in Section 3.5.1.

3.3 The RE project

In this study, the participants were given a set of tasks that involved upgrading the requirements for an existing banking system as represented by its architecture. Their work involved both creating new requirements and evolving old ones in order to create a new requirements set that satisfied the requested changes. For this purpose, they were given the pre-existing requirements and architecture documents (described in the following sub-sections) from the previous version of the system. Each team was given the same 4 requirements-tasks:

- Add *Interac* service to the existing system. It assumes that the transaction is conducted by the bank's employee on behalf of the user. For other services, like Internet banking, this time could be different because of external factors like the user's connection.
- Create a new wireless banking application which would provide features to the customers to carry out basic banking transactions through their cell phones or PDAs.
- Reduce the operational cost of the telephone banking system.
- Increase modifiability in the web banking system.

These tasks were chosen since they constituted a sizeable and complex RE project that would still be feasible within the constraints of a University course. We held numerous peer-review sessions with a total of six experts to validate these four tasks with respect to their appropriateness in giving a project that met both pedagogical and study needs. The requirements elicitation process and techniques followed are described in (Kotonya and Sommerville, 1998).

3.3.1 The pre-existing requirements document

The pre-existing requirements for the system were originally obtained from an external source. These requirements were used to architect the previous version of the system (Ferrari and Madhavji, 2008b). The final requirements from that project are what were used as a baseline requirements set for enhancement in the requirements project on which the study was conducted. Thus, the study project essentially involved one iteration of an *evolutionary cycle* of the system's requirements.

However, these requirements were re-validated by several experts for acceptability in the enhancement project (i.e., the four requirements tasks described earlier). There were approximately 80 requirements in the set, and supporting use cases and sequence diagrams for ten of the key functions of the system. The document structure followed the guidelines from (Kotonya and Sommerville, 1998).

We list here a few example requirements in natural language to give their flavour:

- *The system must complete a transaction in less than three seconds. It that the transaction is conducted by the bank's employee on behalf of the user. For other services, like Internet banking, this time could be different because of external factors like the user's connection.*
- *A customer shall be able to deposit money using ATM into the indicated account by cheque or cash.*
- *A customer shall be provided access to Internet Banking services based on valid bank account number, user defined password, and access permissions set out for the bank customer.*

3.3.2 The architectural document

The architectural document given to each of the RE teams resulted from the described previous study (Ferrari and Madhavji, 2008b). That study involved a set of 16 software architecting teams in an academic setting, each of which worked to create a systems architecture, using the ADD method (Bass et al., 2003). The participants in that study created their architectures based on the requirements mentioned in Section 3.2.1. That study also involved identifying one particular architectural document as being of the highest quality based on an instrument designed for this purpose (Ferrari and Madhavji, 2008b).

The architecture in question was documented in a 161-page document and included information on: quality attribute scenarios, tactics employed, module decomposition views, user/layer views, class views, component and connector views, deployment views, interface specification, work assignment view, sequence diagrams, state charts, and architectural rationale.

3.4 Data collection

In order to gather appropriate data to answer the two research questions, Q1 and Q2 (see Section 3.1), we first designed a meta-model for requirements decisions. Also, to simplify data collection and organization, we developed a software tool based on this meta-model. Furthermore, we had specific measures in place to ensure that quality data would be obtained from this study. These issues are described below in more detail.

3.4.1 The decision meta-model

The decision meta-model specifies the types of entities and relationships involved in the myriad of decisions underlying the requirements process. This meta-model, therefore, can guide data gathering. Since research on requirements decisions is limited, there was no established meta-model available which fitted the specific investigative needs of this study. Instead, a combination of elements from two different sources was used: Ramesh and Jarke's Rationale Submodel (Ramesh and Jarke, 2001) and Wang and Madhavji's Traceability Meta-model (Wang et al., 2005). The integrated meta-model is illustrated in Figure 5-1 and uses UML notation to depict the elements and links.

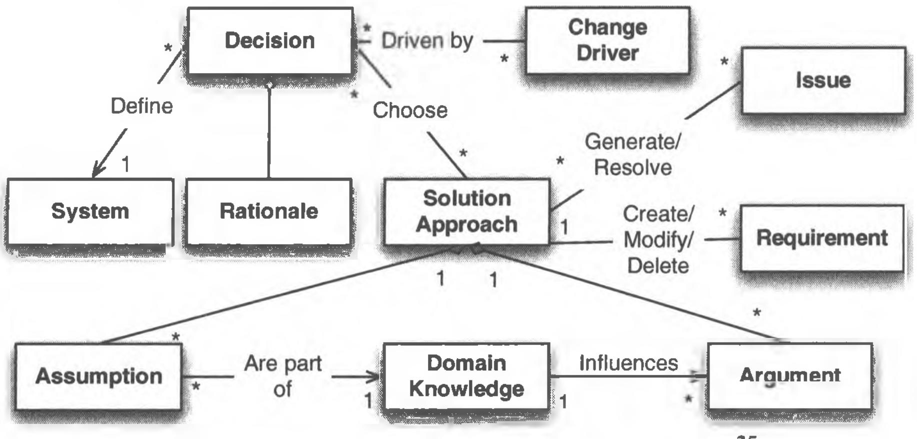


Figure 5-1. A meta-model for RE decisions[35]

The meta-model captures the key notions of decisions, assumptions, requirements and solution approaches. It links various elements to the system through decisions. The input to the model is the Change Driver element. Change Driver is left intentionally abstract since it subsumes many possible drivers of change including (but not limited to) shifting business goals and needs, new contractual requirements, changes in the system's environment, and end-user change requests. In our study, the change drivers were the four project tasks given to the teams (see Section 3.3).

One of the primary attributes that differentiates our meta-model from the earlier ones (Ramesh and Jarke, 2001; Wang et al., 2005) is that, in our model, requirements decisions relate only *indirectly* to requirements, issues and assumptions, through solution approaches. That is, in the ensuing instance-level model (or enactment of the model), each solution approach (i.e., a strategy to meet high level requirements) involves its own set of issues (e.g., cost implications, constraints, actions, etc.), requirements and assumptions (see Figure 5-2). These are only instantiated if the solution approach is *accepted* through a decision (and hence the "indirect" relationship).

[35] Some terms to note: Requirements decision - denotes a chosen subset of high-level requirements (or solution strategies) amongst a set of alternatives in order to achieve a goal; Issue - an important topic or problem for debate or discussion relating to the acceptance/rejection of a solution approach; System – computing system of interest; Rationale – why a requirement is needed with respect to the goals it realizes; Argument – statement supporting or refuting the solution approach; Domain knowledge – the valid knowledge used to refer to an area of human endeavour (in this case the Banking domain); Assumption – a statement which is considered true regarding any aspect of system development.

For example, a decision concerning the reduction of operating costs in the telephone banking system might involve two solution approaches; reducing the number of human operators and/or reducing the available functionality of the system. Both solution approaches are feasible, and the RE team must choose (based on the associated issues) whether or not to implement[36] either of the approaches. Note that it is possible to choose both or neither. Once a decision is made, rationale can be given describing why a particular decision was made (e.g., why a particular solution approach should be implemented over another solution approach).

Specific issues can apply to many solution approaches. Each requirement and assumption is associated to a single solution approach, which can then be traced to one or more decisions. Each requirement has its own rationale, underlying assumptions, relationships to other requirements, importance and other project-related attributes such as cost estimate and tasks. However, these are not elaborated in the meta-model for simplicity. It is around this model[37] that the data collection tool (see Section 3.4.2) was designed.

For each of the elements that help to make up the meta-model, relevant information was captured by the tool. Each element had a unique set of attributes that were captured. The attribute that is of particular interest here is the role that the architecture played in requirements decision-making. The role of the architecture is denoted by whether it acted as an effect (constrained, enabled, influenced, or none) on the requirements decision, and the *aspect* of the architecture that had the effect. The System and Domain Knowledge elements are not directly implemented in the tool, but are meant to provide a context for the rest of the elements and how they fit with the overall system.

[36] Note: though the RE team is not expected to do downstream *development* work (design, coding, testing, etc.), it is evident here that their decision here is carving an implementation path through the "solution approach" they would choose, thereby denoting a problem-solution space relationship.
[37] Prior to the start of the study, peer review with RE experts was used to validate the model's quality.

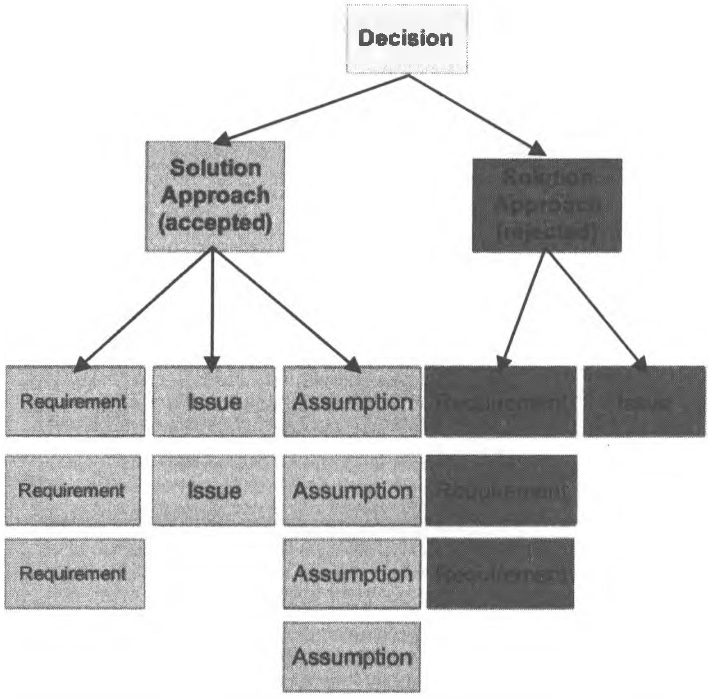
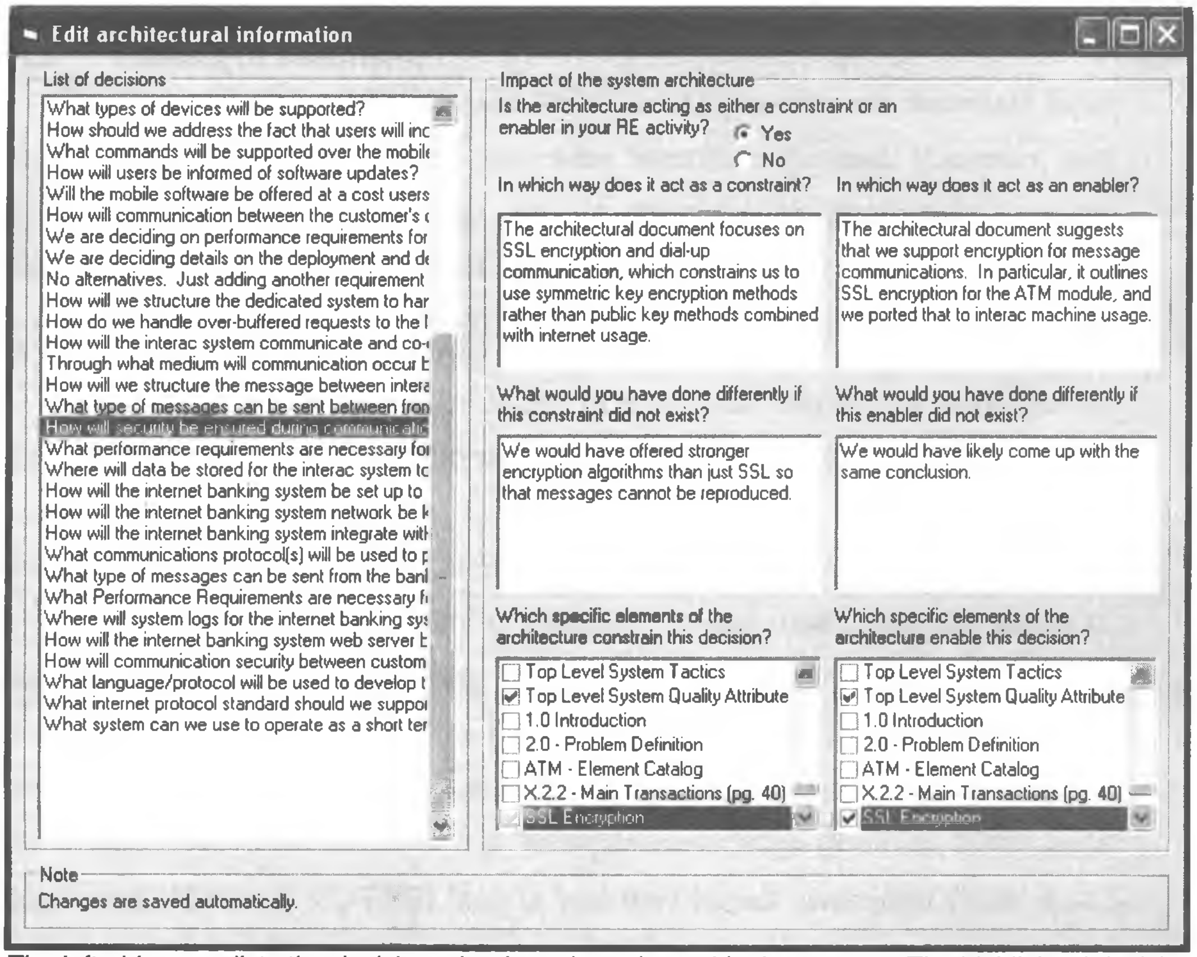


Figure 5-2. A sample decision tree from the meta-model

3.4.2 Data gathering tool

The data-gathering tool could best be described as a *decision-centric requirements engineering tool.* The subjects logged each decision they made into the tool. Each decision had a series of potential solution approaches associated with it, all of which were also logged (see Figure 5-3 for an example screenshot). Underlying the decisions captured, and the way the tool operated is the decision "meta model" described earlier (see Figure 5-1). The tool was implemented in Visual Basic 6 (VB6). It had the dual purpose of supporting the subjects' work and of recording decision data relevant to this study.

Because of this semi-automated tool, data quality could be ensured in several ways that a manual tool (such as forms that subjects must fill out) could not. For example, the subjects could be required to fill in essential fields at the right time such as a requirement's rationale when a requirement is logged so that there's no danger that they might be left blank or, worse, filled in at a later time when the knowledge is no longer fresh. Other fields (e.g., the time of modification) could be generated automatically, thus, alleviating the subjects' workload while guaranteeing correctness of the data.



Edit architectural information

List of decisions

- What types of devices will be supported?
- How should we address the fact that users will inc
- What commands will be supported over the mobile
- How will users be informed of software updates?
- Will the mobile software be offered at a cost users
- How will communication between the customer's c
- We are deciding on performance requirements for
- We are deciding details on the deployment and de
- No alternatives. Just adding another requirement
- How will we structure the dedicated system to har
- How do we handle over-buffered requests to the I
- How will the interac system communicate and co-
- Through what medium will communication occur b
- How will we structure the message between intera
- What type of messages can be sent between fron
- How will security be ensured during communicatio
- What performance requirements are necessary for
- Where will data be stored for the interac system tc
- How will the internet banking system be set up to
- How will the internet banking system network be k
- How will the internet banking system integrate with
- What communications protocol(s) will be used to p
- What type of messages can be sent from the bank
- What Performance Requirements are necessary fi
- Where will system logs for the internet banking sys
- How will the internet banking system web server b
- How will communication security between custom
- What language/protocol will be used to develop t
- What internet protocol standard should we suppor
- What system can we use to operate as a short ter

Impact of the system architecture

Is the architecture acting as either a constraint or an enabler in your RE activity? [✓ Yes] No

In which way does it act as a constraint?	In which way does it act as an enabler?
The architectural document focuses on SSL encryption and dial-up communication, which constrains us to use symmetric key encryption methods rather than public key methods combined with internet usage.	The architectural document suggests that we support encryption for message communications. In particular, it outlines SSL encryption for the ATM module, and we ported that to interac machine usage.
What would you have done differently if this constraint did not exist?	**What would you have done differently if this enabler did not exist?**
We would have offered stronger encryption algorithms than just SSL so that messages cannot be reproduced.	We would have likely come up with the same conclusion.
Which specific elements of the architecture constrain this decision?	**Which specific elements of the architecture enable this decision?**
☐ Top Level System Tactics	☐ Top Level System Tactics
☑ Top Level System Quality Attribute	☑ Top Level System Quality Attribute
☐ 1.0 Introduction	☐ 1.0 Introduction
☐ 2.0 - Problem Definition	☐ 2.0 - Problem Definition
☐ ATM - Element Catalog	☐ ATM - Element Catalog
☐ X.2.2 - Main Transactions (pg. 40)	☐ X.2.2 - Main Transactions (pg. 40)
☑ SSL Encryption	☑ SSL Encryption

Note
Changes are saved automatically.

The left-side pane lists the decisions that have been logged in the system. The highlighted decision is the one being currently worked on. The right-side of the screen is split between two sets of windows: the left-side is where architectural constraint information is logged, and the right-side is where architecture acting as an enabler information is logged.

Figure 5-3. A sample screen shot from the decisions data gathering tool.

3.4.3 Data collection

The data collection phase of this study took place over a span of two months. To help ensure the quality of the data, each team was given one hour a week to meet with a system "stakeholder" played by the course's teaching assistant. During the meetings the subjects had the opportunity to ask questions about the company's needs regarding the new system. Their work to date was reviewed priori to, and discussed at, these meetings to ensure that the subjects properly understood how to use the tool for logging data. Additionally, e-mail communication was used to answer questions regarding tool usage.

3.5 Threats to validity

Based on (Johnson and Christensan, 2004), three types of threats that might apply to the type of study conducted here were identified: External, Construct, and Conclusion validity. Because we are not attempting to demonstrate causality between variables, threats to *internal* validity are not a concern.

3.5.1 Threats to external validity

External validity refers to the degree to which the results of a study can be generalized across a population (Johnson and Christensan, 2004). Threats to external validity occur when researchers draw incorrect conclusions about the population based on the sample data (Creswell, 2003).

Population validity is the ability to generalize the study results from the sample to the population. Because exploratory studies on students have become so prevalent, there is much work done to explore the population validity of students. Specifically regarding SE related student-based studies in academic settings, important results have been found in several cases, e.g., in requirements triage (Runeson, 2003), code inspection (Carver et al., 2003), and in lead-time impact assessment (Host et al., 2000). We do acknowledge the threat in generalizing to experienced requirements engineers and architects; however, there is no evidence suggesting that the results could not be generalizable to, at the very least, novice requirements engineers and architects in industry. Regardless, *exploratory* studies such as this are an important first step towards eventually solidifying a body of knowledge and providing the groundwork for future studies in wider contexts.

3.5.2 Threats to construct validity

Construct validity refers to the extent to which a measurement corresponds to theoretical concepts (constructs) concerning the phenomenon under study. In this study, the constructs (e.g., decisions) were operationalized through the decision meta-model (see Figure 5-1) and the tool that was built on this model. We held numerous peer-review sessions with a total of six experts to validate the meta-model and tool with respect to the theoretical constructs we wanted to investigate (see Section 3.4.1). Also, at no stage in the research process did we come across any instant of data or relationship that questioned the validity of the meta-model or the tool's capability in capturing data pertaining to the meta-model. We are thus confident in the effectiveness of these artefacts for collecting data pertaining to the study's constructs.

3.5.3 Threats to conclusion validity

Conclusion validity is the degree to which conclusions we make based on our findings are reasonable (Trochim, 2006). There are two accepted principles for improving conclusion validity (Trochim, 2006) that applied to our study: ensuring reliability of data measurements and proper implementation of study processes. For reliability of data measurements, we utilized a data-collection tool and weekly meetings to ensure tool was utilized correctly (see Section 3.4.3). Proper implementation was in-place by having a single researcher involved in the study design to perform the various research tasks. Additionally, we discuss the conclusions in the last section of the paper, and there we demonstrate that all our conclusions are rooted in the results, thereby maintaining conclusion validity.

4 Results

This section discusses the findings of the study. We describe first the manner in which the architecture affects requirements decisions (Q1). Then, we describe the quantitative findings related to the specific architectural aspects that affected the requirements decisions (Q2).

4.1 How an architecture affects requirements decision-making (Q1)

The six project teams recorded a total of 117 requirements decisions, all of which related to the four requirements tasks assigned (described in Section 3.3). A significant portion of these decisions was affected in some way by the architecture. We describe the types of effects found in our study and their characteristics.

4.1.1 Types of architectural effects

We identified four types of architectural effects on requirements decisions from our data, shown in Table 5-1 (leftmost column): Enabled, Constrained, Influenced, and Neutral. An architectural effect is of type *enabled* if it makes a solution approach (more) feasible because of the current architectural configuration. Conversely, an architectural effect is of type *constrained* if it makes a solution approach less (or in-) feasible. An *influenced* is where the architectural effect altered a requirements decision without affecting the feasibility of its solution approaches. Finally, the *neutral* type of effect is one where there is no noticeable architectural effect of any kind.

Example 1 -- enabled:
Decision: Implement a back up system for the Interac banking system.
Solution approach 1 (rejected): Introduce new web server which will be used as a backup for Interac transactions.
Solution approach 2 (accepted): Use Internet subsystem web server as a backup for Interac transactions.
Architectural enabler: The web server for Internet already exists. Queue will allow us to hold over 500 transactions and deal with all the requested transactions. Overall System Requirement 1.19 requires that the system should not fail in case of overload.

In this example the decision to use the existing Internet banking web server as a backup for the Interac sub-system was made easier because the team in question knew that existing performance and reliability requirements were sufficient to accommodate the extra workload. This is an example of being enabled by "Non-functional characteristics from a different sub-system" (an aspect of the architecture) than the one being worked upon.

Example 2 -- constrained:
Decision: Establish communications protocol(s) that will be used for the wireless banking system.
Solution Approach 1 (accepted): Communication protocol should be GPRS (General Packet Radio Service).
Solution Approach 2 (rejected): Communication protocol shall be UMTS (Universal Mobile Telecommunications System).
Architectural constraint: Architecture document clearly stated a preference for GPRS; otherwise we would have chosen UMTS.

Here, the example is of a decision being constrained because the decision had already been made in the architectural document.

Example 3 -- influenced:
Decision: Deploy the Interac system.
Solution Approach 1 (accepted): Develop the Interac system based on the conceptual model of system based on the current implementation of the ATM sub-system.
Architectural influence: The architecture document defines functionality for the ATM system. The Interac system can be loosely based upon this conceptual model since the ATM system has been successfully implemented and maintained. Therefore, the presence of the ATM subsystem and how it was implemented influences the solution approach for the Interac system.

This is an example where the decision was not constrained or enabled; nothing about the ATM sub-system makes any of the proposed solution approaches more or less feasible. However, for the sake of consistency, the requirement engineers chose to model the new Interac system after the ATM system. This decision is an example of a decision being influenced by architectural patterns.

Example 4 -- Neutral:
Decision: Determine support for different languages in the mobile banking application.
Solution Approach 1 (rejected): English will be the only language supported.
Solution Approach 2 (accepted): English language as the default language of the system, with other languages to be downloaded and installed on request.
Solution Approach 3 (rejected): Provide support for many languages together with the application.
Architecture Effect (None): The mobile banking application has not been developed, and therefore the technical challenges associated with implementing language support on a wide-variety of mobile devices are considered outside the scope of the current overall system architecture.

Example 4 demonstrates a decision that was unrelated to the existing architecture. In this situation, the mobile banking application has not yet been implemented so the requirement engineers can consider various solution approaches for language support without considering the current architecture.

Note that the described effects are "technical" in nature. That is, our focus is on the "architectural basis" for deciding whether a requirement decision is enabled, constrained, influenced or neutral. In a given software project, there are other factors that also need to be considered in prioritising requirements and in release planning, e.g., implementation cost, revenue potential, and resource requirements. Irrespective of these factors, it is invaluable to know at elicitation-time what the architectural effects are on the decisions being made. Thus, for example, with revenue-potential being equal among two competitive decisions, an enabled decision would be more favourable than a constrained one.

4.1.2 Architectural impact characteristics

Of the 117 requirements decisions mentioned in Table 5-1, 69 of the decisions were affected by the architecture. A decision could be affected by more than one architectural effect (for example, the choice of upgrading a database could be enabled by the current hardware configuration, but also be constrained by poor modifiability in the system components that would need to interact with the database). With reference to Table 5-1, in our study there were 5 such cases, so we had a total of 122 "effect-counts"[38]. Out of the 69 affected decisions, an effect-count of 74 out of 122 (61%) were affected by the architecture. This is a substantial number of effect-counts that were affected in some way. There is, more or less, an even-split between those "Enabled" and "Constrained", which outnumber the category of "Influenced" by a factor of 5.

Type of effect	Architectural Aspects									Decisions
	Existing hardware	NF characteristics (same sub-system)	NF characteristics (different sub-system)	Reusable modules	Architectural patterns	Modifiability	Structural feature	Decision already made	Communication	Number of decisions affected
Enabled	2	4	16	5	3	5	1	4	3	36
Constrained	2	4	8	5	3	3	4	4	6	31
Influenced	0	2	2	0	3	0	0	0	2	7
Neutral										48
Total # of decisions: 117				*Total # of "effect-counts": 122*						

Note that a given decision can be affected by more than one architectural aspect and therefore the number of effect counts may not equal the number of affected decisions.

Table 5-1. Characteristics of architectural impact on requirements decisions

[38] The effect-count includes some decisions in more than one category of effects, thus the summation does not tally, or the % is more than 100.

Equally important is to note that 48 (41%) of the requirements decisions were not affected by the architecture (i.e., type Neutral). Also, all instances of architectural effects on requirements decision-making in our study fit into the defined types of effects.

In previous literature (Shekaran, 1994b), only the "constraint" effect-type was identified. In Section 4.1.1, we identify additional types of effects. Also, in this section, we give quantitative characteristics of the various effect-types. However, it should be noted that different application systems are expected to have different quantitative values because these values depend on factors specific to the development of individual products or systems. Still, it is a subject for future studies as to whether there are approximate quantitative ranges for different effect-types across different applications and application-domains.

4.2 Architectural aspects affecting requirements decisions (Q2)

The types and quantitative characterization (see Table 5-1) of architectural effects on requirements decision-making (Q1) is complemented by the findings of the different aspects of the architecture that had impact on requirements decisions (Q2).

Table 5-1 shows, on the top, 9 architectural aspects that were found to affect requirements decisions in the project. These aspects are:

1. *Existing hardware*: Decisions that were affected by the existing hardware in the system.
2. *Non-functional characteristics (from the same sub-system)*: Decisions that were affected by non-functional characteristics of the *same* sub-system with which the decision was concerned.
3. *Non-functional characteristics (from a different sub-system)*: Decisions that were affected by non-functional characteristics from a *different* sub-system than the one with which the decision was concerned.
4. *Reusability of modules*: Decisions that were affected by the possibility of reusing existing modules.
5. *Architectural patterns*: Decisions that were affected by the choice of architectural patterns already implemented.

6. *Modifiability*: Decisions affected by existing features that were known to be easily modifiable.
7. *Structural features*: Decisions that were affected by structural features of the existing SA.
8. *Decisions already made*: Decisions that were affected when it was realized that the decision in question had already been made in the existing architecture.
9. *Communications*: Decisions that were affected by the existing choice of communications protocols.

Below, we analyze architectural aspects against effect types and against the project groups.

4.2.1 Architectural aspects across effect types

Table 5-1 depicts the role of the architectural aspects (top row) in relation to the type of effects (leftmost column) on the total set of "effect-counts" (122) recorded by the project teams.

Though the category *influenced* occurred less frequently than *enabled* and *constrained*, they are still noteworthy. In our study, *influenced* usually denoted that solution approach used in another part of the system was being used to solve the problem at hand. For example, an architectural pattern might be chosen because it has been implemented successfully elsewhere in the system.

While this may suggest a movement towards a more homogonous architecture, an aspect acting as an influence on *future* RE decisions may be less foreseeable (by a software architect) than those acting as types *enabled* and *constrained*. In particular, whereas *enable*d and *constrained* are related to creating requirements which are consistent with the established architecture and previously made decisions, *influenced* involve implementing previous (or similar) decisions in a new context (i.e., a different part of the system than was originally intended). The risk associated with this, however, is not clear. Thus, if an aspect is known to be of type *influenced*, the architect should be aware that design decisions involving that aspect may have ramifications in other parts of the system that may not be obvious. Care should therefore be taken when architecting these aspects.

4.2.2 Architectural aspects across project groups

Table 5-2 shows the number of requirements decisions that were affected by each architectural aspect and for each of the six project teams. The table shows that the architectural aspect “NF characteristics of a *different* sub-system” affected most number of decisions (20; or 17% of 117 decisions; or 29% of 69 affected decisions)

Besides this, all the remaining architectural aspects affected between 4% and 13% of the affected decisions (see last row in Table 5-2). Also, we see that in Table 5-1, the aspect “NF characteristics (different sub-system)” has the greatest % of “enabled” requirements decisions (16 of 36, or 44%).

	Architectural Aspects									Decisions		
Team	Existing hardware	NF characteristics (same-subsystem)	(different sub-system)	Reusable modules	patterns	Modifiability	Structural feature	made	Communication	# Affected	# Total	% Affected
1	0	0	0	0	0	2	0	0	2	4	12	33
2	1	0	0	0	0	1	1	0	1	4	12	33
3	0	0	3	1	2	0	0	0	0	6	16	38
4	2	3	10	1	4	2	3	3	4	32	38	84
5	0	1	7	4	0	1	0	3	0	16	24	67
6	0	5	0	0	0	0	0	0	2	7	15	47
#Total	3	9	20	6	6	6	4	6	9	69	117	59
% out of total (117)	3	8	17	5	5	5	3	5	8			
% out of affected (69)	4	13	29	9	9	9	6	9	13			

Table 5-2. The relationship between architectural aspects and project teams.

We do see some discrepancies, however. While “NF characteristics (different sub-systems)” was the most active architectural aspect (see Table 5-2), the instances of affected requirements decisions came from groups 3, 4 and 5. One explanation for this phenomenon could be that this particular aspect depended on how much effort the subjects put into understanding sub-systems that were non-local to those in their focus of attention. Indeed, while acquiring an understanding of the other sub-systems in the architecture did actually affect the decision making of groups 3, 4 and 5, it is possible that the other groups simply did not focus their attention on seemingly unrelated sections of the architectural documentation. We do not have data for this analysis, and so future empirical studies could help explain this phenomenon.

Despite this variance between teams, we include all data points since this a multiple case study. However, including this data results in 59% of requirements decisions being affected by architectural aspects (as seen in Table 5-2, last column, 3rd row from the bottom), while their exclusion would result in 52% of the decisions being affected, so there is not much difference. Thus, we will simply state that the architectural aspects listed affected approximately 50% of the requirements decisions.

5 Implications

There are a number of implications for SA and RE of the findings from our study:

Planning and risk management: The analysis and categorisation, during the RE process, of architectural effects on RE decisions (see Section 4.1.1.) could help architects to separate the more easily implementable, enabled, requirements from the more difficult to implement (or compromised), constrained, requirements. This separation of concerns could be useful from the point of view of project planning (e.g., time-to-implement, resource allocation, requirements prioritisation and scheduling), risk management (e.g., implementability) (Boehm, 1988), and product evolution (e.g., new feature planning).

For example, one group in the dataset elicited high-level requirements to reduce the cost of telephone operators in telephone banking by introducing an automatic speech recognition system. These requirements were "enabled" in two principle ways: one, by readily available COTS systems/components from the marketplace and two, by the modifiability of the current implementation of the telephone sub-system. The same group elicited high-level requirements for the mobile banking application, specifically that the existing infrastructure (i.e., servers and their throughput) could be used to handle the mobile banking application transaction load. However, these requirements were assessed as "constrained" because of the existing performance demands from the other major types of access to the system (e.g., Internet, teller, etc.). So, for planning purposes, the management had to decide: Should I upgrade the SA in order to implement the requirements for the mobile application, which has a potentially high positive impact on the customer's point of view? Or, should I implement instead the requirements for the automated phone system, where these are less desirable from the customer point of view but less-time consuming to implement and hence can lead to releasing the system faster and thus start saving money by removing the human telephone operators?

RE and SA technology: Similarly, this separation of concerns of architectural effects could help researchers and tool developers to enrich the requirements elicitation and analysis tools (e.g., DOORS, Requisite pro, *i** (Liu and Yu, 2001), etc.) which, in turn, could enrich SA tools (e.g., ArchE (Diaz-Pace et al., 2008), Software Architect, etc.) in making judicious choices of architectural tactics and patterns to satisfy quality requirements. Currently, RE and SA tools do not consider the presence of an existing system when performing further RE and SA work, and therefore do not facilitate the presentation or analysis of information describing the RE and SA interaction effects (such as the information in Table 5-1). Integrating this information, and subsequent analysis support, into RE and SA interaction tools could then enable users to make decisions based on information that is currently left implicit.

Likewise, this separation of concerns can help in implementing, automatic, dialogue-triggering mechanisms in RE-to-SA workflow processes (Georgakopoulos et al., 1995), especially for the "constraint" category of requirements. That is, the RE and SA agents can be notified automatically to resolve the tradeoffs between implementing a constrained decision (at the expense of customer dissatisfaction) and implementing an unconstrained decision (at the expense of architectural modifications).

Architectural evolution: Historical trends of aggregate quantitative data (as in Table 5-1) can aid in SA management and in opportunistic or restrained RE practice. For example, if the trend shows that too many RE decisions are constrained by the specific aspects of the legacy SA (e.g., 8, 5 or 6 in the "*Constrained*" row in Table 5-1) then this might call for: (i) examination of SA practices and developing checklists to ensure that architects are not inadvertently restricting potential future business goals; (ii) restructuring[39] the SA to align it with business goals; or (iii) restraining the RE process (from attempting to integrate unconstrained requirements into the constraining parts of the SA) until such time that the architecture has been adequately restructured. Conversely, trends of too many enabled decisions (e.g., 16 in the "*Enabled*" row for "NF characteristics (different sub-system)" in Table 5-1) could possibly indicate that the enabling aspects of the SA are, at least, technologically supportive of the new ventures and can unleash RE to be more opportunistic. This type of analysis and questioning is not a part of architecting methods (e.g., ADD (Bass et al., 2003), GRL (Liu and Yu, 2001), and CBSP (Egyed et al., 2001)) or architecture evolution approaches (e.g., ArchWare (Morrison et al., 2007), ESDM (Shen and Madhavji, 2006), FIESTA (Wagnier et al., 2007)), and, doing so, could allow for improved architectural evolution support.

Tighter SA-RE integration: With over 50% of the RE decisions being affected by an SA (see Table 5-2), and many of these (29% or 20/69) originating from the aspect "NF characteristics of a different subsystem", this is strong empirical evidence in favour of integrating software architecting and RE processes more tightly (Nuseibeh, 2001). Specifically, the SA agents could work with the RE agents during requirements elicitation, negotiation and feasibility analysis in order to provide critical insight on the technical feasibility of the elicited requirements in terms of them being *constrained* or *enabled* from a *different* sub-system as opposed to the subsystem they are working on.

[39] SA restructuring can include such tasks as: capability analysis (of the SA as to whether it can cope with stakeholder scenarios), tactics and pattern choices, technology assessment, deployment strategies, and others.

Therefore, a hypothesis emerges (see Section 6) that, in order to reduce the amount of backtracking and requirements-rework (and also reduce the associated project costs), it is important that the architects provide "live" feedback to the RE agents on these potential system-wide "constraints" and "enablers".

However, due to resource constraints in RE and SA processes of a software project (for example, in some projects it may not be possible for requirements engineers to have extensive interaction with the architects), at the very least requirements engineers could analyse different sub-systems than the one they are working on to possibly discover more local requirements decisions that could be *enabled*. If this is so, requirements engineers could be trained, and appropriate tools developed, specifically for this circumspective analysis in order to yield more enabled solutions for better service and satisfaction to the end user. As mentioned in the introduction section of this paper, the current industry practice does not align with this recommendation.

RE to SA feed-forward process: Iterative development approaches (such as RUP [Kruchten, 2001] and Spiral [Boehm, 1988]) tend to promote that significant chunks of requirements are validated and prioritised preceding the development effort. While this may be quite appropriate in many situations, there is room to be agile in some situations across RE-architecting processes by introducing "feed-forward" processes from RE to SA. In particular, requirements engineers can package critical information and deliver this to the architects *prior* to the delivery of the validated new requirements. For example, in our case-study projects the requirements engineers could have packaged information about the four architectural categories of high impact (see Section 4.1.2: *existing hardware, NF standards (same sub-system), NF characteristics (different sub-system), and architectural patterns*), the specific requirement decisions that are affected, and how they were affected (e.g., constrained, enabled or influenced). This package of information, if made available to the architects "ahead of time", could facilitate groundwork for specific architectural enhancements, and change, while the rest of new requirements are still being elicited in the RE process. We note that agile practices (Larman, 2003) do not explicitly promote such feed-forward processes from user stories to system development.

Increased middleware: The *neutral* type of effect has a significant amount of cases (approx. 40%, see Table 5-1). Neutral cases actually mean that the developers will likely have to "wire in" the design and code for a new requirement into the system much more deeply than in the "enabled" cases where, for example, the groundwork would already have been prepared in the existing architecture for the new requirements to be implemented. Deeper the "wiring in", higher the software costs in general and more arduous the development. Thus, some of the "wiring-in" work could possibly be reduced in the future by increased "middleware" strategy in the architectural design.

Analysis: So, as we see above, there are quite a few implications of determining architectural effects on requirements decisions: on early software development practices, methods and tools. The identified implications are threads for further empirical work to ground them in development processes.

6 Future empirical work

One purpose of an exploratory study is to lay a foundation for possible future work on the theme of the research so as to build an appropriate body of knowledge (IEEE SWEBOK, 2004). In a sense, the exploratory study is conducted in a "bottom-up" manner, where the research question acts as a guide to collecting a wide range of data about the research topic, and the findings are discovered from the exploratory analysis of this data. In an effort to lay such a foundation, it is important to identify any emergent hypotheses or investigative questions from this research. From such hypotheses, it would then be possible to conduct, in a "top-down" manner, quantitative studies that focus on specific research issues. The main purpose of conducting a "top-down" study is to statistically test the hypothesis to lend quantitative support to the topic being investigated.

From the results of our study and their implications, below we describe the following four emergent hypotheses that could be tested in future studies and how they could be tested:

Hypothesis 1: *If the architects provide "live" feedback to the RE agents on potential system-wide constraints and enablers, then the amount of requirements-rework will be reduced.*

See Section 5: *Tighter SA-RE integration* section for a more detailed discussion of the background of this hypothesis. To test this hypothesis, we would need to measure the amount of requirements rework between two different groups of Software Engineers. This measurement could include the amount of requirements-rework needed to be done, and also the extent of the rework (i.e., effort and time). One of these study groups would have requirements engineers and architects who are working together in an integrated manner to develop the requirements and architecture; the other type of group would not have the requirements engineers and architects working as closely integrated. For this hypothesis, the independent variable would be the requirements and architecting process used, and the dependent variable is the time and effort expended performing requirements-rework.

Hypothesis 2: *Non-functional (NF) characteristics of a non-local sub-system significantly affect (enable or constrain) requirements for the local sub-system being worked on.*

In Table 5-1, we see that NF characteristics of a different sub-system than the one being worked upon affected requirements more than any other aspect. This could have potentially important implications on RE and SA technology as discussed in Section 5. Despite this importance, prior to investigating into new technologies, there is a need to replicate this study in different domains and contexts in order to determine generalizability.

To test this hypothesis, therefore, two types of RE and SA groups are needed for the study: one that is given the entire architecture including information regarding the NF-characteristics of all the subsystems; whereas, the other group does not receive this NF information. Both groups would elicit requirements for a single subsystem and, as in this study, architectural aspect analysis is performed and the number of impacted requirements is logged and statistically compared. The independent variable would then be the presence/absence of NF-characteristic information of non-local subsystems, and the dependent variable would be the reported number of impacted requirements.

Hypothesis 3: *If the history of interaction effects between SA and RE is used effectively, then the time/effort spent performing evolutionary work in requirements and architecture processes will decrease.*

As discussed in Section 5, maintaining and using the history of information presented in Table 5-1 could be useful for evolutionary work in the requirements and architecting processes. This hypothesis aims at providing scientific evidence as to whether or not having such information is useful and, if so, to what extent.

A controlled experiment involving two study groups could be used to test this hypothesis. Development teams expected to enhance a system (both requirements and architecture) would be used. One type of study group would be given the historical interaction effect information from the past revisions of the system; whereas, the other group would not receive this information. Process data such as effort and time would be gathered and then analysed to determine any statistically significant differences between the two types of groups. The independent variable is the historical information, and the dependent variable is the time and effort spent in performing an evolutionary phase in an RE and SA project.

Hypothesis 4: *Architectural communication protocols used in the current system have a significant effect on new requirements.*

In Table 5-1 in Section 4.2, communication protocols used in the architecture have an effect on new requirements. Despite the finding that the effect is mostly *constrained*, this is more likely due to a function of our product circumstances, thus we generalize this hypothesis for all the types of effects. Establishing further evidence of this claim can lead to improved RE and SA technology where this issue is more explicitly considered in those processes.

To test this hypothesis, a study with two types of RE groups enhancing the requirements for an existing system is needed. One type of group will be given an existing system architecture with fully realized communication protocols. The other type of group would be given an existing architecture, however, the communication protocols would be undetermined. The two types of groups would provide data on the architectural aspects affecting the requirements they are eliciting, and in the end the number of requirements affected by communication protocols would be statistically compared to determine evidence to support or refute the hypothesis. The independent variable is the realization of communication protocols, and the dependent variable would be the reported number of affected requirements.

7 Conclusions

The role of an existing systems architecture (SA) in requirements engineering (RE) was recognised as important over a decade ago (Shekaran, 1994a). However, to our knowledge, this issue has not been scientifically explored. This paper describes an exploratory study on this question. This study involved six RE teams eliciting requirements to enhance an existing system, and collecting and analyzing data from their in-project decisions that they made. Collection of data was facilitated by a tool that allowed the teams to not only do their requirements work but also capture study-specific data. This tool was based on a requirements decision meta-model (see Figure 5-1) that was designed and validated for use in this study.

From the findings of the study, we conclude that:

1. There exist at least four types of architectural effects on RE decisions (see Section 4.1.1): as an enabler (30%), as a constraint (25%), as an influence (6%), and as neutral (39%). This means that approximately 60% of the RE-decisions were affected (or approximately 40% were not affected) by the SA. These characteristics add significant new knowledge to the literature (Shekaran, 1994b) where the existence of the "constraint" effect was suspected but the different types of effects and their extent were not known.

2. Also, different aspects of the SA can have different degrees of effects on RE decisions (see Section 4.1.2). From our study, there were nine different aspects of which "non-functional characteristics (of sub-systems other than the one the analyst is working on for eliciting new requirements)" had the most impact on the affected RE decisions: approximately 29%.

There are several implications of the findings on: Planning and Risk management; RE and SA technology; Architecture evolution; SA and RE processes; and Middleware. These are discussed in Section 5.

Apart from the general need to replicate empirical studies, several notable suggestions for future empirical work would be to conduct studies based on the following four emergent hypotheses: (1) *architects providing "live" feedback to RE agents on system-wide constraints and enables will reduce amount of requirements-rework,* (2) *Non-functional characteristics of non-local sub-system significantly affect requirements for the local sub-system being worked on,* (3) *time/effort spent performing evolutionary work in requirements and architecting processes will decrease if history of interaction effects between SA and RE is used effectively,* and (4) *architectural communication protocols used in a current system has a significant effect on new requirements.*

Since ours was only one exploratory-based study in a particular context, it would be a mistake to generalise these results verbatim to other contexts (Zave, 1997). However, this does not diminish the importance of the findings described in this paper. Instead, we encourage the readers to view this study as an important first step for establishing grounded theory for future studies in this area.

Acknowledgement
This work was, in part, supported by Natural Science and Engineering Research Council (NSERC) of Canada.

References

Baldwin, C. Y. and Clark, K. B., 2000. Design Rules, Vol. 1: The Power of Modularity. The MIT Press.

Bass, L., Clements, P., Kazman, R., 2003. *Software Architecture in Practice*, Addison-Wesley.

Boehm, B., 1988. A spiral model of software development and enhancement. IEEE Comp. Vol. 21, Iss. 5, pp. 61-72.

B. Boehm and V. Basili, 2001. Software Defect Reduction Top 10 List. IEEE Computer, vol. 34(1): 135-137, January 2001.

Hans de Bruin, Hans van Vliet, 2003. Quality-driven software architecture composition. Journal of Systems and Software 66(3): 269-284.

Carver, J.; Shull, F.; Basili, V., 2003. Observational Studies to Accelerate Process Experience in Classroom Studies: An Evaluation. Proc. of the 2003 Int. Symp. on Emp. Software Engineering (ISESE '03), Rome, Italy, pp. 72-79.

Creswell, J. W., 2003. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* Thousand Oaks, CA: Sage Publications.

Xiaofeng Cui; Yanchun Sun; Hong Mei, 2008. Towards Automated Solution Synthesis and Rationale Capture in Decision-Centric Architecture Design. Seventh Working IEEE/IFIP Conference on Software Architecture, Feb. 2008 Page(s): 221 – 230.

Diaz-Pace, Andres; Kim, Hyunwoo; Bass, Len; Bianco, Philip; & Bachmann, Felix, 2008. Integrating Quality Attribute Reasoning Frameworks in the ArchE Design Assistant. Proceedings QoSA'08, 4th International Conference on the Quality of Software Architecture. University of Karlsruhe (TH), Germany. October 14-17.

Egyed, A., Grunbacher, P., Medvidovic, N., 2001. Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach. First International Workshop from Software Requirements to Architectures (STRAW '01), Toronto, Canada, June, 2001.

El Emam, K.; Madhavji, N. H., 1995. Measuring the Success of Requirements Engineering Processes. Proc. of the 2nd IEEE International Symposium on RE, York, England, March 1995, pp. 204-211.

Rik Farenhorst, Patricia Lago and Hans Van Vliet, 2007. EAGLE: Effective Tool Support for Sharing Architectural Knowledge. International Journal of Cooperative Information Systems, Vol. 16, Nos. 3 & 4, pp. 413–437.

Ferrari, R. and Madhavji, N. H., 2008a. Architecting-problems rooted in requirements. Information and Software Technology, Volume 50, Issue 1-2 (January 2008), Pages 53-66.

Ferrari, R. and Madhavji, N. H., 2008b. Software architecting without requirements knowledge and experience: What are the repercussions?. Journal of Systems and Software, Volume 81 , Issue 9 (September 2008), Pages 1470-1490.

Garlan, D., 1994. The Role of Software Architecture in Requirements Engineering. Proceedings of the First International Conference on Requirements Engineering, April, 1994, pp. 240.

Georgakopoulos, D., Hornick, M. and Amit Sheth, 1995. An overview of workflow management: From process modeling to workflow automation infrastructure. Journal of Distributed and Parallel Databases, Volume 3, Number 2, April 1995, pp. 119-153.

C. Hofmeister, R. Nord, D. Soni, 2005. Global Analysis: Moving from Software Requirements Specification to Structural Views of the Software Architecture. IEEE Proceedings Software, Vol. 152, Issue 4, pp. 187-197, August 2005.

M. Host, B. Regnell, C. Wohlin, 2000. Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering, pp. 201–214.

Guide to the Software Engineering Body of Knowledge: 2004 Version. IEEE and IEEE Computer Society project. <http://www.swebok.org/>.

Jackson, M., 1994. The Role of Architecture in Requirements Engineering. Proceedings of the First International Conference on Requirements Engineering, April, 1994, pp. 241.

Johnson, R. B., and Christensan, L., 2004. Educational Research: Quantitative, Qualitative and Mixed Approaches. Allyn & Bacon; 2 edition.

Kazman, R., Klein, M., Clements, P., 2000. ATAM: Method for Architecture Evaluation. Technical Report, Software Engineering Institute, Carnegie Melon University, CMU/SEI-2000-TR-004 ESC-TR-2000-004.

Thorsten Keuler, Dirk Muthig, Takayuki Uchida, 2008. Efficient Quality Impact Analyses for Iterative Architecture Construction. pp.19-28, Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008).

Kotonya, G.; Sommerville, I., 1998. *Requirements Engineering*. John Wiley & Sons Ltd.

Kozaczynski, W., 2002. Requirements, Architectures and Risks. Proceedings of the IEEE Joint International Conference on Requirements Engineering, Essen, Germany, pp. 6-7.

Kruchten, P., 2001. The Rational Unified Process: An Introduction. Second Edition. Addison-Wesley, Boston.

Matthew J. LaMantia, Yuanfang Cai, Alan MacCormack, John Rusnak, 2008. Analyzing the Evolution of Large-Scale Software Systems Using Design Structure Matrices and Design Rule Theory: Two Exploratory Cases. pp.83-92, Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008).

Larman, Craig, 2003. *Agile and Iterative Development: A Manager's Guide.* Addison-Wesley Professional, August 21.

Liu, L. and Yu, Eric, 2001. From Requirements to Architectural Design – Using Goals and Scenarios. 2nd Int. Workshop from Soft. Reqts. to Arch. (STRAW '01), Toronto, Canada.

Miller, J., Ferrari, R., Madhavji N. H., 2008. Architectural Effects on Requirements Decisions: An Exploratory Study. 7th Working IEEE/IFIP Conference on Software Architecture (WICSA '08), Vancouver, Canada, pp. 231-240.

Miller, J., Ferrari, R., Madhavji, N. H., 2009. Characteristics of New Requirements in the Presence or Absence of an Existing System Architecture. Proceedings of the 17th IEEE Conference on Requirements Engineering (RE '09), Atlanta, United States, August 2009.

Ron Morrison, Dharini Balasubramaniam, Flavio Oquendo, Brian Warboys, and R. Mark Greenwood, 2007. FIESTA: A Generic Framework for Integrating New Functionalities into Software Architectures, First European Conference on Software Architecture (ECSA 2007), LNCS 4758, pp. 2 – 10.

Nuseibeh, B., 2001. Weaving Together Requirements and Architectures. IEEE Comp., March 2001, 34(3): 115-117.

Nuseibeh, B.; Easterbrook, S., 2000. *Requirements engineering: a roadmap.* Proceedings of the Conference on the Future of Software Engineering, ACM Press, pp. 35-46.

Ramesh, B. and Jarke, M., 2001. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, Volume 2, Issue 1, pp. 58-93, January 2001.

Rapanotti, L., Hall, G., Jackson, M., Nuseibeh, B., 2004. Architecture-driven problem decomposition. In: Proceedings of the 12th IEEE International Requirements Engineering Conference (RE 2004), Kyoto, Japan, pp. 80–89.

Runeson, P., 2003. Using Students as Experiment Subjects – An Analysis on Graduate and Freshman Student Data. EASE'03 – Proc. 7th Int. Conf. on Empirical Assessment & Evaluation in Software Engineering, April, 2003, pp.95-102.

Schwanke, R., 2005. GEAR: a good enough architectural requirements process. In: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, USA, pp. 57–66.

Shaw, M., 2003. Writing good software engineering research papers: minitutorial. In: Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), Portland, USA, Tutorial Session, pp. 726–736.

Shekaran, C., 1994a. Panel Overview: The Role of Software Architecture in Requirements Engineering. Proceedings Of the First International Conference on Requirements Engineering, April, 1994, pp. 239.

Shekaran, C., 1994b. The Role of Software Architecture in Requirements Engineering. Proceedings of the First International Conference on Requirements Engineering, April, 1994, pp. 245.

Shen, Y., and Madhavji, N. H., 2005. ESDM – A Method for Developing Evolutionary Scenarios for Analysing the Impact of Historical Changes on Architectural Elements. 22nd IEEE International Conference on Software Maintenance (ICSM'06), pp. 45-54.

Stoll, P.; Wall, A.; Norstrom, C., 2008. Guiding Architectural Decisions with the Influencing Factors Method. Seventh Working IEEE/IFIP Conference on Software Architecture, Feb. 2008 Page(s):179 – 188.

Software Requirements to Architectures Workshop (STRAW), 2001. International Conference on Software Engineering (ICSE) workshop, June 2001, Toronto, Canada.

Software Requirements to Architectures Workshop (STRAW), 2003. International Conference on Software Engineering (ICSE) workshop, May 2003, Portland, USA.

Tichy, W.F., Lukowicz, Prechelt, L., Ernst, A., 1995. Experimental evaluation in computer science: a quantitative study. Journal of Systems and Software (January), 1–18.

Trochim, W.„ *2006. Research Methods Knowledge Base.* This is available at http://www.socialresearchmethods.net/kb/design.php. Last accessed January 2009.

Vogt, P., 1993. *Dictionary of Statistics and Methodology: A Nontechnical Guide for the Social Sciences.* Sage Publications, California, US.

Waignier, G., Anne-Franc □Loise Le Meur, and Laurence Duchien, 2007. FIESTA: A Generic Framework for Integrating New Functionalities into Software Architectures. First European Conference on Software Architecture (ECSA 2007), LNCS 4758, pp. 76–91.

Wang, Z, Sherdil, K., Madhavji, N., 2005. ACCA: An Architecture-centric Concern Analysis Method. IEEE Working International Conference on Software Architecture (WICSA), Pittsburgh, USA, November 2005, pp. 99-108.

Wieringa, R.J., Heerkens, J., 2006. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. Requirements Engineering Journal 11, 295–307.

Zave, P., 1997. Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys, Vol. 29, No. 4, pp.315-321.

Chapter 6

Requirements Engineering Decisions in the Context of an Existing Architecture: A Case Study of a Prototypical Project[40]

1 *Introduction*

In (Miller et al., 2008), we describe a laboratory-scale study on the effects of the existing systems architecture (SA) on new requirements engineering (RE) decisions – those typically in the context of an evolving system. The basic issue of interest in that study was whether the RE decisions are influenced in any way if the existing systems architecture is taken into account in the RE process. This issue is important because RE *does* take place in the context of existing SA in 60-80% of software development (Huff, 1990), if not more, and yet there are virtually no scientific studies on this matter and the Software Engineering Body of Knowledge (SWEBOK) (IEEE SWEBOK, 2004) or the RE standard (IEEE Std., 1998) are practically devoid of relevant advice.

For example, what proportion of the newly elicited requirements is likely to be constrained by the system's architecture? Or, what is the time and effort impact of SA-constrained requirements, as opposed to non-constrained requirements, on downstream development?

Not knowing answers to such questions implies: (i) RE practice is *ad hoc* in terms of how to treat SA when engineering requirements; (ii) pedagogical literature is silent on this issue and thus learning and training is held back; and (iii) research advances on RE methods, tools and processes are oblivious of the "what, why and wherefore" of SA in RE which, in turn, is holding back to some extent learning, training and disciplined practice of RE.

[40] A version of this chapter was published in (Ferrari et al., 2010).

As a matter of fact, the issue of the role of a SA in RE has been brewing in the RE research community since the mid-90s (Jackson, 1994; Nuseibeh and Easterbrook, 2000), albeit with little progress to date. In our laboratory study (Miller et al., 2008), we identified four types of architectural effects – and their extent -- on RE decisions: (i) *constrained* (25%) -- the existing SA makes a solution approach (i.e., high-level requirements) less (or in-) feasible; (ii) *enabled* (30%) -- the existing SA makes a solution approach (more) feasible; (iii) *influenced* (6%) -- the architectural effect alters a requirements decision without affecting the feasibility of its solution approaches; and (iv) *no effect* (39%) -- the architecture has no known effect on a requirements solution.

While these initial findings are promising, it is generally accepted in the wider scientific community that such laboratory studies are only a starting point in developing a grounded body of knowledge and that there are compelling reasons for conducting further empirical studies, including replications (Neuliep, 1991), involving real-world projects (Kitchenham et al., 2004). This is critical none more so than in the field of Software Engineering (SE) in general (Brooks et al., 2008), and RE in particular (Wieringa and Heerkens, 2006), because they are still considered in their infancy. In this respect, the recent first international workshop on replication in SE is noteworthy (RESER, 2010).

In this paper, we present results of a case study on a large-scale prototypical, automated, rail project (RailCab) being carried out in Germany. These results support the previous findings (Miller et al., 2008) in terms of the different types of SA effects on RE decisions (see above) though there is some variance in the extent of the effects, which is to be expected from a radically different type of project (RailCab is systems engineering[41] in automated trains; whereas, [Miller et al., 2008] was a banking system) and scale (RailCab is a 10-year, real, pre-production, prototype train involving professionals; whereas, [Miller et al., 2008] was a class project over one term involving students).

[41] In this paper, the focus is on the RailCab system as a whole, which includes both hardware and software aspects. Thus, we do not separate "software" parts from "hardware" parts.

This paper also describes completely new findings, not investigated in (Miller et al., 2008), on two major research questions: (1) the characteristics of the RE decisions affected by SA, based on the requirements evolutionary framework described in [8] and the type of requirements identified by Sommerville (Kotonya and Sommerville, 1998), and (2) the impact of the affected RE decisions on (i) downstream development activities (such as construction and testing) in terms of time and effort and (ii) the RailCab system as a product. These findings have implications for tighter RE and SA integration across subsystems, SA impact analysis, project planning and risk management, and future empirical research in RE based on three emergent hypotheses, which are also described in this paper.

The case study involved an investigation of the 10-year history of requirements and architecting decisions in several major components of RailCab (drive and brake, energy management and active guidance). The data was collected from numerous project documents and extensive interviews with the RailCab developers and planners. In total, 108 requirements decisions were examined.

In the next section we describe the case study design; in Section 3 we present the results of the case study; Section 4 discusses example implications; in Section 5 we provide a summary and comparison to related work; and lastly, Section 6 concludes the paper.

2 The Case Study

In this section, we describe the core parts of the case study. This includes: the research questions, an overview of the RailCab project, case study participants, data collection and analysis procedures, and threats to the study.

2.1 Research Questions

We have three pertinent research questions:

Q1: What is the impact of an existing system's architecture on RE decisions?

This question replicates the investigation in (Miller et al., 2008) on the impact the presence of an architecture has on decision-making in RE. Requirements decision-making leads from recognition of a problem to be solved to a specification of that problem or a solution strategy (Miller et al., 2008), which is in contrast to an architectural decision that deals with the structure of the system in terms of the key structural elements of the system, and their interrelationships (Garlan, 2000). Basically, a RE decision denotes a chosen subset of high-level requirements (or solution *strategies*) amongst a set of alternatives in order to achieve a goal. For example, deciding to provide a web-based self-help service to clients (as opposed to phone-in service or personal contact service) in order to cut down operational costs. It is through the choice of such high-level business strategies that detailed requirements are then elicited and established. This decision-making process is not strictly a top-down process. For example, detailed requirements for several strategies may first be elicited and assessed (for relative business advantage, feasibility, cost, resource consumption, etc.) prior to deciding upon a particular subset of strategies to implement (Nuseibeh, 2001). Thus, an individual requirement is only "indirectly" related to a RE decision through identified *strategies* (Miller et al., 2008). Nonetheless, requirements are explicitly traceable, at one end, from more abstract constructs such as RE decisions, strategies and scenarios and, at the other end, from software artefacts such as lower-level design, code and test cases (Ramesh and Jarke, 2001). This notion of requirements decisions is operationalised through the decision meta-model designed and validated in (Miller et al., 2008).

We list here an example requirement decision, D1, from the RailCab project and, for D1, we give alternate strategies:

D1: To use Nickel cadmium batteries for the Energy Management Component during system development and testing.

Solution strategy 1 (accepted): Nickel cadmium batteries are robust, safe, and relatively easier to implement. Thus, they will be used during the experimental development and testing of the system; however, because they are costly, they will not be used for the operation of the final system.

Solution strategy 2 (rejected): Nickel-metal hydrate batteries weigh less and take up less physical space than Nickel cadmium batteries, allowing for more batteries to be installed in the same physical space while increasing the maximum energy capacity. Furthermore, they are cheaper than Nickel cadmium batteries. However, they are more difficult to implement and therefore impose more requirements on the system, thus they will be used for final operation of the system, and not during system development and testing.

In deciding whether to elect solution approach 1 or 2, factors to be considered include, SA constraints (or impact on the RE decision), cost, time and system functionality and quality. Both strategies were constrained by the physical space in the RailCab, but this was more the case with the first strategy. However, the time for implementing strategy 2 is considerably higher, so the decision was to accept strategy 1 for the development and testing of the system, but not for the final system. The second strategy will be used for the final system only. Note that decision D1 relates to a solution strategy in that it lays out a plan for the types of battery to use for the Energy Management Component. By itself, D1 is not an "architectural" decision because it does not indicate how the batteries are to be "structurally" organised (i.e., patterns) as part of the overall system architecture and how these batteries will be interacting with other modules of the system (i.e., interfaces). Thus, decision D1 is essentially the first high-level step towards eliciting more detailed requirements and, hence, it is deemed a requirement decision and not an architectural decision.

The research question Q1 is investigated by collecting and analyzing the data from two constructs: the *requirements decisions* (such as D1 above) and their *RE and SA interaction type* (i.e., whether a decision such as D1 is constrained, enabled, or influenced by the SA, or is neutral – see Section 1).

Furthermore, this paper surpasses the limits of analysis in (Miller et al., 2008) in two significant ways: (a) the characteristics of the affected decisions are determined (e.g., source of the decision), and (b), the impact of the affected decisions on the system and process (e.g., implementation and testing) is identified. These new results are consequences of the following two new research questions posed in this paper:

Q2: What are the characteristics of the affected decisions?

In so far as "requirements" are concerned, literature has long promoted the idea of categorising them. For example, (Harker et al., 1993) has categorised requirements by source (e.g., external stakeholder request, fix an unforeseen implementation problem, requirements that emerge during detailed planning, etc.) and Sommerville (Kotonya and Sommerville, 1998) has categorised requirements into type (e.g., non-functional, functional, deployment, etc.). Although in the case study we are dealing with RE "decisions" and not requirements *per se*, we still used the above categories because of several reasons: (i) requirements decisions are closely related to requirements themselves, (ii) one can examine a requirements decision in conjunction with its relevance to the project or business goals and associate appropriate categories to that decision. Such categorisation of RE decisions can aid analysis in software projects.

Q3: What is the impact of the affected RE decisions on the resultant system and downstream development activities?

Researchers have investigated the impact of RE practices on downstream activities (Damian and Chisan, 2006), and while these studies were indeed interesting, they did not scrutinize such impact in the context of constrained architectural effects on RE decisions. Such an understanding has the potential to influence project planning, traceability, and RE and SA interaction technology. Note that question Q3 is examined from two angles: product and process. For each affected decision, we interviewed the RailCab project staff to qualitatively determine the impact of RE decisions on (i) the system and (ii) activities outside of the requirements elicitation process, in particular: requirements costing, implementation and testing processes, system reliability, safety, and maintainability.

2.2 Study Context: The RailCab project

The RailCab project has been in development for approximately ten years at the University of Paderborn in Germany, with a budget of over twelve million Euros. The project is expected to continue for several more years. The train's test track is approximately 530 metres in length with one track-switch and one railway station, and the RailCab vehicles are constructed to the scale of 1:2.5. The goal of the prototype project is to introduce new technologies that can be used in future production rail systems. RailCab is considered a "mechatronic" system, i.e., it requires the interdisciplinary expertise in the areas of mechanical, electrical and software engineering fields.

2.2.1 Features and Components

The key feature of the RailCab[42] vehicle is that it is an autonomous, self-optimizing system, and thus does not require any human operator to drive the train. The RailCab consists of five major components: Drive and Brake, Energy Management, Active Guidance and Steering, Tilt and Suspension, and Track Topology and Motor Design. The first four components contain a mix of hardware and software components, where the software plays the role of embedded controllers within the hardware.

Figure 6-1 depicts a high-level architecture diagram of the RailCab[43], with the major components shown as rectangles and the key dependencies between them represented as lines with arrow connectors. For example, in this diagram it can be seen that the Energy Management Module is one of the core modules of the system, as all the modules depend on its operation. Conversely, the Active Guidance and Tilt/Suspension operate on top of the Drive and Brake and Energy Management, and thus are not essential for safe operation of the RailCab. More description of the modules investigated in this study is given in Section 2.4.

[42] Readers are encouraged to view videos showing the RailCab executing at different stages of its development. The videos can be viewed at http://nbp-www.upb.de/index.php?id=57&L=1

[43] Note that this diagram intentionally does not depict the complex information and control flow that exist between the modules in order to provide a simple high-level diagram of the architecture. For more information regarding the technical SA details of the RailCab, the readers are referred to the many publications associated with the RailCab project at

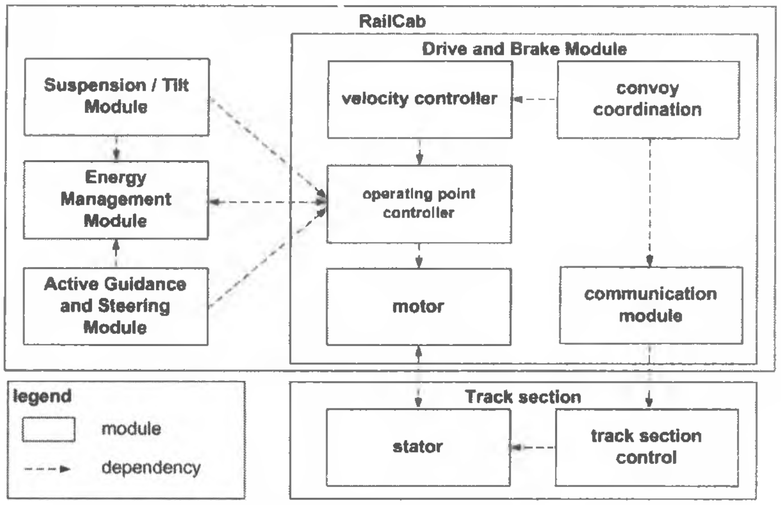


Figure 6-1. High-level RailCab architecture

2.2.2 Prototypical development process

The RailCab's development process is understandably quite different from that of a "traditional" production project. Its process is exploratory and iterative; the system is continually evolving in different directions depending on the current research ideas being investigated. This is in stark contrast to a production environment which is more a planned iterative or waterfall process. The research process also implies a tighter and fixed budget that constrains the developers to find partial solutions that will work given their limited resources. Conversely, in a production project, the project costs often greatly exceed the original budget due to hardware replacement, extensive re-work, and business priorities. In the research process, there are fewer developers allocated to the project, and a combination of full-time employees, university professors, research assistants and students are employed resulting in a higher turnover in staff than that in a production project. Another implication of the prototypical process is that the collection and reporting of process data is much "looser" than in a production project where these issues would be formalized and more strictly enforced. Lastly, in the RailCab project, external standards and regulations do apply but are almost strictly related to the development lab itself (e.g., development and testing of the RailCab does not emit excessive gases into the environment) and thus these standards do not have serious repercussions on

http://www.sfb614.de/en/sfb614/subprojects/project-area-d/subproject-d2/

the RailCab development. In contrast, in a full production system meant for travelling passengers and cargo there will be a dramatically large number of regulations and standards imposed from external sources such as the government and engineering standards.

2.2.3 Requirements process

The core set of features in the RailCab project originally came from senior members of the staff and also external partners (e.g., industry partners, government, etc.). Beyond these core features, any member of the RailCab staff can elicit new requirements to implement in the project; the main driver for a new requirement is the research innovation that it brings and as such, the sources of the requirements were often cutting-edge technology from research literature that could be applied to the RailCab project. However, new requirements that have an impact across more than one major module, or require hardware purchases, must be approved by the senior members of the staff and communicated to the entire staff. Part of the approval process is the prioritisation of new requirements, based mainly on the innovation that a new requirement brings counterbalanced by cost implications. Requirements that are deemed to be low in priority will not be implemented on the RailCab vehicles themselves but can still be implemented in the laboratory (i.e., small standalone testable units or simulations).

2.3 Participants

In this study, eight senior-level developers and researchers were extensively interviewed over a span of approximately one year on a bi-weekly basis for approximately 1-2 hours each interview session. Additionally, they provided project documents and validated emergent findings from the study. Each developer is primarily responsible for his/her own major module. They each have over five years of experience and have expertise in systems engineering, specifically in electrical, software, control and mechanical engineering. Their primary project tasks are the implementation and deployment of the system; however, they are also key project members in the front-end systems development activities such as RE and SA.

2.4 RailCab Modules Investigated

In this study, three of the five major modules of the RailCab were investigated: Energy Management, Drive and Brake, and Active Guidance. The other two modules were omitted because the primary planners and implementers of these components were not accessible for the extensive interviews that were conducted as part of this study. We now provide a general description of each of the three investigated modules.

The primary purpose of Energy Management Component is to ensure that each of the RailCab subsystem's energy demands are fulfilled. Additionally, the module is responsible for recharging the energy sources (through its innovative hybrid energy system) as the RailCab vehicle operates. Other features include heat and voltage monitoring of battery arrangements for safety purposes, using batteries as main power supply for driving if track energy is not available, and adjusting energy levels at runtime based on differing priority levels of subsystems requesting energy.

The Drive and Braking module is responsible for the general autonomous driving and braking of the vehicle and also includes features such as forming convoys of multiple individual vehicles, and maintaining routes and speeds that avoid possible unsafe driving states. The module uses physical sensors that collect data from the environment (such as pertinent track information like slope and track quality, or obstacle detection) and from this data the RailCab vehicle automatically determines the speed and acceleration of the vehicle.

The Active Guidance module is primarily responsible for the smooth lateral motion of the vehicle. Whereas the Drive and Brake module is responsible for basic driving parameters such as speed and acceleration, the Active Guidance is in charge of optimizing the lateral motion and steering of the vehicle to promote smoothness of the vehicle, in particular when traversing around corners in the track.

All of these components involve a mix of hardware and software components. The software is executed on an on-board computer in the RailCab vehicle and controls the various functions of the components listed above.

2.5 Data Collection and analysis

The case study involves examining each major module of RailCab; specifically, the history of the requirements and architecting decisions in the project, but also the interaction effects between newer requirements elicited in the presence of the existing SA. There are numerous qualitative-based sources of data for this investigation that include: meeting minutes, planning documents, theses and technical reports, research papers, presentation slides, prototypes, and other project documents such as memos, notes and bulletins.

In addition to these documents, the other primary source of data is the RE and SA knowledge from the RailCab developers (see Section 2.3) elicited through semi-structured interviews. These interviews were audio recorded (in excess of 25 hours) and subsequently transcribed (into over 275 pages of typed text) to provide a written account of the interviews.

Since the data collected is mostly qualitative, analysis techniques more commonly associated with the Social Sciences were used. Specifically, *content analysis* (Creswell, 2003) was used to analyse the project documents and interview text. In short, the technique is when the researcher scans through the textual data and categorizes text segments of interest. In our case, this was annotating any text pertaining to requirements decisions and their impact from the existing SA. This technique was supported by the qualitative analysis tool Nvivo 8[44], which facilitated the storage of annotations, allowed for creation of links between different textual sources, and facilitated the creation of tables, charts and matrices to visualize computed frequencies of the categorization.

2.6 Threats to Validity

From (Runeson and Host, 2009), three types of threats that might apply to the case study proposed here were identified: External and Construct validity, and Reliability. Because we are not attempting to demonstrate causality between variables, threats to *internal* validity are not considered.

[44] QSR NVIVO 8. QSR International Pty Ltd., 2010. Available at http://www.qsrinternational.com.

2.7.1 Threats to External Validity

External validity refers to the degree to which the results of a study can be generalized across a population, time and setting (Runeson and Host, 2009).

Ecological validity refers to the generalizability of the study results across all settings. As discussed in section 4.1, RailCab is a prototypical project that is carried out in an experimental setting. Furthermore, the domain is systems-oriented and its primary drivers are safety-critical and real-time performance. These project and system characteristics are different from those of the banking system in our previous study (Miller et al., 2008) so comparisons across the domains need to be made prior to any generalization and beyond this, generalization should not be taken for granted. However, there is hope that that the results would be useful, if not completely generalizable, to other prototypical projects in the systems domain.

2.7.2 Threats to Construct Validity

Construct validity refers to the extent to which a measurement corresponds to theoretical concepts (constructs) concerning the phenomenon under study. In this study, the constructs (e.g., requirements decisions) were operationalized through the decision meta-model designed and validated in (Miller et al., 2008). The data itself comes directly from the project employees and documents. Additionally, numerous researchers external to the project validated the results and interpretations to ensure that that the constructs are properly measured.

2.7.3 Reliability

Reliability is concerned with the extent that the data and analysis are dependent on the specific researchers of the study (Runeson and Host, 2009). To contain this threat, data triangulation was used with the study's multiples sources of data (see Section 2.5) used to corroborate data from the interview sessions with participants. Researcher triangulation was also used, where one other researcher and the study participants reviewed and validated the study's analysis and results, as well as the audio transcript files (see Section 2.5) derived from the interview sessions in which they were involved.

3 Results

We now describe the results and interpretations from the data analysis that was performed for the investigation of the three research questions (see Section 2.1).

3.1 Architectural Impact on RE decision-making (Q1)

In the three major modules investigated (see Section 2.4), a total of 106 requirements decisions were extracted from the project documents and interviews with the RailCab staff. A substantial portion of these decisions was affected in some way by the evolving architecture. Here, we describe the characteristics of these RE and SA interactions.

Overall, 37 out of the 106 decisions (35%) were affected by previous architectural decisions. Likewise, this implies that 69 out of the 106 decisions (65%) were *not* affected. Out of the 37 affected decisions, 25 were of the type *constrained (*23%*)* and 13 were of type *enabled* (12%).

These figures are slightly different to what we observed in our previous case study from the banking domain (Miller et al., 2008). Overall, approximately 55% of decisions were affected, meaning 45% were not affected. Out of the affected decisions, the *constrained* vs. *enabled* was 30% vs. 23%. Likewise, unlike in our previous case study (Miller et al., 2008), there were no observations of the effect type *influenced* (i.e., if the architectural effect altered a requirements decision without affecting the feasibility of its solution approaches).

When examining the distribution from the three investigated components, the Energy Management component had the highest number of affected decisions at 47%, followed by the Drive and Brake and Active Guidance at each approximately 32%. When interviewing the developers further about this discrepancy, they indicated that much of the functionality of the latter two components was software-driven and this software needed to be written from scratch and was done independently of earlier major systems and software architecting decisions that were made. On the other hand, the Energy Management Component's architecture had three properties that lead to it having tighter association with SA decisions: (1) its architecture was tightly coupled with the initial decisions on the Track Topology and

Motor Design, essentially the "load-bearing" (Garlan, 2000) decisions of the entire RailCab vehicle, (2), its functionality is critical to the overall running of the system, as it provides energy to all the major functional modules of the system, leading to interdependencies with every major module of the system, and (3), the component is very hardware-focused, which means there is inflexibility in the changes that can be made and thus potentially leads to more *constrained* decisions.

3.2 Characteristics of Decisions (Q2)

We now probe into the characteristics, justified in Section 2.1, of the different types of decisions (i.e., constrained, enabled, neutral) – when posing question Q2. The characteristics of interest are the "source" of the requirements decision and, for this, we use the categories from a requirements evolution framework in (Harker et al., 2003), and the "type" of requirements, given in (Kotonya and Sommerville, 1998). The associated categories are: (i) *consequential*, new (or modified) requirements decisions triggered by feedback from implementation activities, (ii) *core* or *stable*, decisions that are essential for the system, (iii) *emergent*, decisions which cannot be (or may not have been) completely defined when the system is specified but which emerge as the system is designed and implemented (e.g., some UI requirements), (iv) *functional* based decisions, (v) *non-functional* based decisions, and (vi) *implementation* decisions. The key idea is to determine how many RE decisions, and of what type, fall under which categories – as depicted in Table 6-1. This would give us a handle on reasoning about the various decisions.

Decision Categories	**Type of SA effect**		
	Constrained	**Enabled**	**Neutral**
Consequential	18 (72%)	4 (31%)	10 (14%)
Core/Stable	3 (12%)	6 (46%)	34 (49%)
Emergent	4 (16%)	3 (23%)	26 (37%)
Total	*25*	*13*	*70*
Functional	11 (44%)	9 (69%)	42 (60%)
Non-functional	10 (40%)	4 (31%)	23 (33%)
Implementation	4 (16%)	0 (0%)	5 (7%)

Table 6-1. Characteristics of requirement types.

Constrained:

A fairly substantive number of new RE decisions were constrained by previous architectural decisions (see Table 6-1). 18 of the 25 (72%) constrained decisions were classified as *consequential*; decisions that emerged as a consequence of implementing other requirements decisions, often triggered by feedback from other implementation-based development activities. This is a substantial number and a key finding here is that consequential decisions would be constrained and should thus be treated with a tighter RE and SA integration for compatibility reasons (see *Impact analysis* in Section IV for further discussion of the implication of this finding). The fact that *consequential* decisions were the most *constrained* may be intuitively obvious, however, the extent and the relative frequency to the other categories is new knowledge and currently not reported in the literature.

Of these 18 consequential decisions, 5 (28%) were resultant from architectural oversights made previously in *other* components that were not discovered until the implementation and testing phases of development. The domain experts considered these decisions as the most problematic, because they were unknowingly constrained by previous SA decisions in other components where these experts were not involved in the RE and SA decision-making. Furthermore, because of the inter-disciplinary nature of the RailCab project, the domain expertise required for each different module can be quite different, further leading to problems when dealing with architectural oversights from other components.

For example, a requirements decision in the Energy Management component was *to use the onboard power supply as a backup power source only when the public energy network failed.* This decision turned out to be constrained by the physical space in the track switch segment of the track; the energy transformers that were installed in the other segments of the track could not be installed properly in the track segment portion because of a previous architectural decision to have a "reaction" rail which would help manually guide the RailCab vehicle through the track switch. However, the physical space of the reaction rail was overestimated, leading to problems in this portion of the track. This previous faulty architectural decision could not be fixed without extensive cost and development time expenditure, so the solution was to come up with a new solution approach *to always use the onboard power supply when driving through a track switch segment.* Accepting this alternative approach meant that the lower capacity energy of the onboard supply resulted in advanced features of the RailCab (such as Active Guidance optimizing processes) being disabled during driving through the switch, ultimately resulting in a negative impact on stakeholder satisfaction. See *Tighter SA-RE integration* in Section IV for a more detailed discussion of the implication of this finding.

The remaining 13 out of the 18 (72%) were mostly self-contained decisions within a single module, and were not considered as problematic as the previously discussed 5 decisions.

The remaining 7 (28%) decisions were *core* (i.e., essential for the operation of the RailCab) or *emergent*. The high number of *consequential* decisions, coupled with the low frequency of *core* and *emergent,* suggest a trend that the source of *constrained* decisions can be characterized as almost "work-around" decisions in response to implementation of previous requirements that did not go as smoothly as planned.

Neutral:

We now discuss the *Neutral* column in Table 6-1, acknowledging that this is out of order of the table but instead presenting the results in order of interest. These decisions can be characterized as mostly stable or core decisions (34 out of 70 – 49%), or emergent during development (26 out of 66 – 39%). Basically, these decisions were predominantly made during the early phases of planning of the RailCab, which spanned approximately 2-3 years, and remained stable for the entire duration of the development process – therefore did not suffer being consequential as much as the constrained category. Furthermore, these decisions and their subsequent implementation solutions were largely dictated by the system's domain (automated trains) and in many cases did not offer many alternative solution strategies (e.g., decisions regarding energy converters to use, how the converters will be structurally connected to the various units requiring energy, using an accelerometer to measure vertical acceleration, etc.).

Enabled:

As shown in Table 6-1, the results for the *enabled* decisions do not show any discernible trend towards a bias in any of the categories; there is an almost split between all the categories. This suggests that the source of enabled requirements can come from any source.

Requirements Type:

In Table 6-1, we see an even ratio of functional and non-functional decisions being affected in all the types of categories (*constrained, enabled, neutral*), suggesting that there is no bias towards the type of the decision being affected by the existing SA. For the *implementation* type, the frequency count in the RailCab data set was too low to discern any noticeable trends.

3.3 The Impact of Affected Decisions (Q3)

With reference to question Q3, we qualitatively probed the affected decisions to determine the impact (positive or negative) they had on development activities (e.g., construction, lower-level design, testing, etc.) and the resultant system. These results are rooted in the interviews with the RailCab's development staff (see Section 2.3) and are based on their accounts of the events and issues with each affected decision (compared to their perception of events and issues had there been no effect).

3.3.1 Impact on Development Processes

Referring to Table 6-2 (development activities), we can clearly see that constrained RE decisions had a noticeable impact on activities outside the RE process[45]. In particular, the top three development activities that suffered increased time and effort due to constrained RE decisions were *construction* (i.e., hardware implementation), *testing* (i.e., testing process, including actual testing of system as well as design of test cases) and *systems architecting*. The measures are in terms of RE decision counts.

[45] We could not report on actual cost and time impact since this data was not reliably recorded during the execution of the project.

For construction, 20 out of the 25 (80%) constrained RE decisions resulted in increased hardware assembly and software coding time and effort. For testing, 18 out of the 25 (72%) cases resulted in extra time and effort in creating new test cases as well as testing procedures. For systems architecting, 10 out of the 25 (40%) cases resulted in extra time and effort in re-architecting the physical space and layout and determining which hardware to purchase/design. For the other development activities (e.g., software architecting, pure software implementation, etc.), impact was also observed but in no more than 5 cases. An obvious question that surfaces is, "How do we know that the constrained RE decision *increased* development time and effort"? This judgment is based on stakeholders' unanimous opinions.

Affected		**Constrained RE decisions (25 Total)**
Development Activities (20 decisions)	Systems Architecting	10 (38%)
	Software Architecting	5 (19%)
	Construction	20 (77%)
	Software Implementation	5 (19%)
	Testing	18 (69%)
	Other	4 (15%)
Product (17 decisions)	Physical design	4 (24%)
	Modifiability	2 (12%)
	Reliability	3 (18%)
	Availability	2 (12%)
	Driving performance	2 (12%)
	Loss of functionality	2 (12%)
	Other	4 (24%)
# of overlapping decisions (Activities and Product)		12

Table 6-2. Impact of constrained RE decisions on process and system as per data from interviews.

For example, to implement one constrained RE decision (*Improve the smoothness and comfort of the RailCab driving through Active Guidance self-optimization processes)*, the domain experts reported that they had to: (i) replace some of the existing sensors (costing over 100, 000 Euros), (ii) physically re-organize and attach the new sensors because they could not be installed in the same way as the former sensors, (iii) re-implement the software interfaces and connectors that interact with the new sensors, (iv) change the data formats that the sensor data could receive and, (v) all the above changes had to be thoroughly tested to ensure that the desired result was achieved. The extra time and cost in these downstream activities to implement the RE decision was thus directly attributed by the domain experts to the previous SA decisions that were implemented, namely, the physical architecture of the undercarriage and the sensors that were used.

For the *enabled* cases, the impact of RE decisions on other activities is difficult to discern since there are no counter-cases to compare against. In these cases, the benefit could only be observed during RE decision making; but later, the implementation, lower-level design and testing was not reported any differently than not affected decisions.

3.3.2 Impact on System

Not only development activities were affected by constrained RE decisions, the resultant system quality (such as modifiability, reliability, availability and performance) was also affected negatively, though slightly less as measured by decision counts – see Table 6-2: 17 out of 25 (68%) for system vs. 20 out of 25 (80%) for activities). 12 of the 17 (70%) decisions also affected downstream activities, implying that these were hybrid process/product decisions. In most of these hybrid decisions, however, the reported degradation of system quality was characterised by the developers as "slight". That is, much time and effort was spent in downstream activities ensuring that the threat to system quality was mitigated. There was not a single *constrained* decision where no impact on downstream activities or system was reported.

However, in a couple of cases, the resultant product quality was substantially less than originally desired, in addition to increased construction and testing time. For example, in one decision in the Active Guidance module the desired quality is that the Active Guidance is always available, barring any problems in other critical modules (such as the Energy Management Component). However, the physical architecture of the track topology, in particular the track switches, did not allow for power transfer and thus the steering of the RailCab vehicle had to be aided manually with a track-locking device that guided the RailCab.

As with the previous subsection, it was difficult to discern any positive benefit in the *enabled* cases because they followed a similar implementation path as the *not affected* cases.

These results seem to fit the characteristics of a prototypical project where, in most constrained decisions, developers could not simply upgrade or replace hardware components due to the cost involved. Instead, they had to spend a lot of time and effort in finding alternative solutions that still provided near-desired levels of system quality.

4 Implications

There are a number of implications for SA and RE of the findings from our study. We discuss four examples here:

Tighter SA-RE integration across different subsystems: With 35% of the RE decisions being affected by the architecture (see Section 3.1), and several of the affected decisions originating outside of the component being analysed (see Section 3.2 – *Constrained* for the specific results), it is strongly encouraged that the SA and RE processes be more tightly integrated (Nuseibeh, 2001). This corroborates with our earlier findings from (Miller et al., 2008), where 29% of the affected decisions originated from non-functional properties of non-local subsystems.

Specifically, the RE agents should work with the SA agents (not only those responsible for the areas/subsystems RE agents are working upon but also those responsible for other areas/subsystems) during requirements elicitation, negotiation and feasibility analysis in order to provide critical insight on the technical feasibility of the elicited requirements in terms of constraints and enablers from a *non-local* subsystem as opposed to the subsystem they are working on.

In RailCab, RE and SA decisions are predominantly made synonymously within a single subsystem and no distinction is made between SA and RE roles. However, the iterative approach used in RailCab contributed to a separation between RE and SA concerns when working across the different subsystems. For example, some of the early decisions for the motor and track topology architecture led to constraints in the energy management system which the planners knew about but deferred until later. RE and SA were highly intertwined in the motor and track subsystem, yet during this early planning phase the focus was almost entirely on the motor and track subsystem; high-level requirements were elicited for the energy subsystem but no detailed RE or SA work was done at that time. After the motor and track architecting phases were near completion, the energy subsystem's detailed RE and SA phases commenced. However, it was then determined that previously known constraints would be more difficult to plan and implement because of tradeoffs introduced in architectural decisions from the motor and track subsystem. Thus, one lesson learnt from this is that during the architecting phase of motor and track, corresponding detailed RE and SA work should also have been carried out in the energy subsystem to handle alignment issues.

Impact analysis of requirements on SA: Literature promotes that the impact of new requirements on the existing SA is analysed during the RE process, irrespective of the source of these requirements (Jackson, 1994; Nuseibeh, 2001). However, this analysis can be costly, especially when done at a detailed level of all architectural components, as architects, requirements analysts and other appropriate stakeholders need to communicate and coordinate on this task. While our study results show that implementation of *constrained* RE decisions was costly (see examples in Section 3.3), the overall findings from Section 3.1 show that approximately 65% of the RE decisions were not affected by the existing SA, and therefore did not require detailed SA impact analysis. This, empirical, finding contravenes the informal wisdom promoted in the literature (Nuseibeh, 2001). Thus, this raises an important issue, i.e., can we *a priori* identify classes of requirements that are more or less likely to be affected by the SA?

The characterization of affected RE decisions based on source and type in Section 3.2 is a first step towards such identification. We observed two discernible trends in this section that can aid in planning for SA impact analysis. The first trend is that approx. 70% of the *constrained* RE decisions (18 out of 25, see section 3.2 - Table 6-1) were triggered by feedback from implementation activities (i.e., *consequential* decisions), suggesting that when requirements analysts receive requirements from this source, they really ought to perform detailed SA impact analysis as recommended in the literature (Nuseibeh, 2001). However, approx. 85% of the *neutral* RE decisions (i.e., not affected), were predominantly *core* or *emergent* decisions (33+26 = 59 out of 69 – see Table 6-1 in Section 3.2) which suggests that requirements coming from the domain, or that which can be classified as core requirements for a system, will be absorbed more easily into the existing SA, prompting less time and resources expended performing SA impact analysis during RE.

Planning and Risk management: In the RailCab project, as we saw in the example of Section 2.1, that management had to choose between: (1) re-architecting and reconstructing the physical layout of the RailCab vehicle, to allow the use of nickel-metal hydrate batteries which facilitated faster development time and (2) implementing the safety requirements associated with the nickel-cadmium batteries, where these requirements are less desirable in the short-term (with regard to providing innovative features for demonstrative purposes) but provide better system quality for the long-term. To help in this decision-making, the analysis and categorisation, during the RE process, of architectural effects on RE decisions (see Section 3.1) can help architects to separate the more easily implementable, enabled, requirements from the more difficult to implement (or compromised), constrained, requirements.

Further empirical work in RE: Based on the findings of this case study, we raise the following example emergent hypotheses:

H1: Significantly more consequential requirements affect the SA than do core or emergent requirements.

This hypothesis emerges from the finding in Section 3.2, where it was shown that *consequential* decisions were often *constrained* by the SA (70%), and the *core* requirements decisions were more often not affected by the SA.

H2: Older, "load-bearing" components of a system lead to more constrained effects on new requirements decisions than newer implemented components.

This hypothesis comes from the detailed explanation of the results in Section 3.1, where it was observed that the SA components causing the constraints for new requirements decisions were the first components to be implemented (in the RailCab project, these were implemented approximately 4-6 years prior to new decisions in other components), and they were also the core components of the system (i.e., many components depended on them).

H3: The implementation of constrained requirements decisions has a more negative impact on downstream development activities than on product quality.

This hypothesis emerges from the findings in section 3.3, where it was shown *constrained* requirements decisions were more problematic in terms of effort, time, and rework on downstream development activities than on product quality. The experts reported that for most cases, despite the constraint from the SA, they could still achieve (near) desired product quality.

5 Related Work

In (Miller et al., 2008), we discuss at length related work pertaining to the role of a software architect in the RE process. In particular, there we describe three key aspects: (i) relationship between RE and SA based on observations and empirical work, (ii) technological research spanning RE and SA, and (iii) recent technological-based research on architecture evolution. Here, however, because this is a paper on, in part, a replication of the previous study (albeit in a much more sophisticated context), we do not re-describe all that related work but, instead, summarize it below and make links to specific experiences in the RailCab project.

In 1994, Jackson (Jackson, 1994) gave some key reasons why RE and SA are best treated as interweaving processes, e.g.: (i) creating an abstract design as a way of better understanding the system's specifications; (ii) assessing alternate architectures as a way of creating specifications that can be economically implemented; and (iii) ease of movement of developers within the bounds of requirements and architecture. As described previously (see Section 2.2), the RailCab project followed a fluid, prototype, development process with overlapping sub-process boundaries (e.g., between requirements, architecture, coding and testing) in order to experiment with innovative ideas and assess implementation risks. All the key points described by Jackson (Jackson, 1994) seem very relevant for the RailCab's development.

In 1995, El-Emam and Madhavji (El Emam and Madhavji, 1995) found four factors for RE success in information systems that deal with architecture and/or the system. One of these factors is relevant for this study: the *adequacy of diagnosis* of the existing system (which includes SA). We saw in Section 3.1) that existing system implementation in the RailCab project had significant impact on new RE decisions, either as constraints (25%) or enabled (12%). Also, from Table 6-1 we saw that 70% of the constrained RE decisions were "consequential", meaning that these decisions and associated requirements emerged as a consequence of implementing other requirements decisions, often triggered by feedback from other implementation-based development activities. Furthermore, in section 3.3 we saw how the constrained RE-decisions in RailCab led to rework during implementation and increase in cost and time. The lack of significant system diagnosis during the elicitation of requirements in the RailCab project – and the ensuing consequences -- seems to lend support to the finding for RE success by El-Emam and Madhavji.

In 2001, Nuseibeh (Nuseibeh, 2001) described the "twin-peaks" model, which captures the iterative relationship between RE and SA. An important aspect of this model is that SA can, and should, feed back into the RE process (as well as having RE feed into SA, as usual). This aspect is discussed in more detail under the "tighter RE and SA integration" implication in Section 4.

In (Rapanotti et al., 2004), Rapanotti et al., propose the extension of "problem-frames" (a structured way of describing the problem space, pioneered by Jackson [Jackson, 2001]) into "architecture frames", which capture information about architectural styles and their interaction with the problem space. The benefit of this mechanism is that in introducing solution-oriented approaches early in development, one can refine problem analysis. We can clearly see the link between this interesting work on problem and architecture frames and the empirical study on SA effects on RE decision in the RailCab project as well as in (Miller et al., 2008). What is more, the RailCab study has led to the characterisation of RE decisions (such as: consequential, code/stable, emergent, etc. – see section 3.2) and the consequences on rework, cost and time due to constrained RE decisions. Perhaps, the problem and architecture frame approaches can be extended to capture such empirical results for the benefit of evolving projects.

While these are some of the key works highlighting the role of SA in RE[46], the body of knowledge on this topic is fairly thin overall and has basically remained static.

6 Conclusions

In this paper, we describe the impact an existing Systems Architecture has on requirements decisions, determined through a case study on a large-scale, prototypical, rail project (RailCab). This study is an extended replication of an initial exploratory study (Miller et al., 2008) that was conducted in a "laboratory" setting. The case study involved the analysis of approximately 10 years worth of project documents and extensive interviews with RailCab staff – with a focus on three of the five major RailCab system components (Energy Management, Active Guidance, and Drive and Brake). In a nutshell, we found 106 requirements decisions where: 37 (35%) were affected by a previous architectural decision; 26 (25%) of these decisions were *constrained* by the existing architecture and 13 (12%) of these decisions were *enabled* by the existing architecture (see Section 3.1). These results are comparable to that found in the previous study (Miller et al., 2008) and provide a critical step in the empirical process by generating comparable results across domains and study contexts, that can lead to a more solidified body of knowledge in SE (Kitchenham et al., 2004).

Further to the above overall results, this study probed deeper into the affected decisions in two significant ways: (i) to characterize the decisions based on their type

[46] We exclude the work focused on transitioning "from RE to SA", which abound in the literature, because predominantly these works do not explicitly consider the effects of SA on RE decisions.

(Kitchenham et al., 2004) and source (Harker et al., 1993), and, (ii), to qualitatively determine their impact on downstream development activities and properties of the resultant system. For (i), we found that approximately 70% of the constrained decisions originated from feedback in downstream development activities (see Section 3.2), and the not-affected decisions had the characteristic as being core or domain-driven decisions (86%) (see Section 3.2). In (ii), it was observed that all constrained decisions had a negative impact on product or process, with the severity being higher on the process side (see Section 3.3).

These results have implications for: project planning and risk management, tighter RE and SA integration across subsystems, and SA impact analysis. Also, three emergent hypotheses from this study (see Section 4) form the basis for future empirical research in RE.

Acknowledgment
This work was, in part, supported by Natural Science and Engineering Research Council (NSERC) of Canada. This contribution was also developed and published in the course of the Collaborative Research Center 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" funded by the German Research Foundation (DFG) under grant number SFB 614.

References

Brooks, A., Roper, M., Wood, M., Daly, J., Miller, J., 2008. Replication's Role in Software Engineering. Guide to Advanced Emp. SE, Springer London, pp. 365-379.

Creswell, J. W., 2003. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Thousand Oaks, CA: Sage Publications.

Damian, D., Chisan, J., 2006. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. Trans. on SE, 32 (7), 433–453.

El Emam, K., Madhavji, N. H., 1995. Measuring the Success of Requirements Engineering Processes. Proceedings of the 2nd IEEE International Symposium on RE, York, England, March 1995, pp. 204-211.

Ferrari, R., Sudmann, O., Geisler, J., Henke, C., Schaefer, W., Madhavji, N. H., "Requirements Engineering Decisions in the Context of an Existing Architecture: A Case Study of a Prototypical Project", In Proceedings of the 18th IEEE Requirements Engineering Conference, Sydney, Australia, 2010, pp. 79-88

Garlan, D., 2000. Software Architecture: a roadmap. Proceedings of the Conference on the Future of Software Engineering, ACM Press, pp. 91-101.

Gausemeier, J.; Schäfer, W.; Greenyer, J.; Kahl, S.; Pook, S.; Rieke, 2009. Management of Cross-Domain Model Consistency during the Development of Advanced Mechatronic Systems. Proceedings of the 17th International Conference on Engineering Design (ICED'09), pp. 1-12, Stanford, USA.

Giese, H., Tichy, M., Schilling, D., 2004. Compositional Hazard Analysis of UML Components and Deployment Models. Proceedings of the 23rd International Conference on Computer Safety, Reliability and Security, pp. 166-179, Potsdam, Germany.

Harker, S., Eason, K, Dobson, J.E., 1993. The change and evolution of requirements as a challenge to the practice of software engineering. IEEE International Symposium on Requirements Engineering, pp. 266–272.

Huff, S., 1990. Information systems maintenance. The Business Quarterly 55, 30-32.

IEEE's Guide to the Software Engineering Body of Knowledge: 2004 Version. IEEE and IEEE Computer Society project. http://www.swebok.org/

IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, 1998.

Jackson, M., 1994. The Role of Architecture in RE. Proceedings of the First International Conference on Requirements Engineering, pp. 241.

Jackson, M., 2001. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley.

Kitchenham, B., A., Dybya, T., Jøorgensen, M., 2004. Evidence-based software engineering. In ICSE '04: Proceedings of the 26th International Conference on Software Engineering., pages 273–281, Washington, USA.

Kotonya, G., Sommerville, I., 1998. Requirements Engineering: Processes and Techniques. Wiley.

Miller, J., Ferrari, R., Madhavji N. H., 2008. Architectural Effects on Requirements Decisions: An Exploratory Study. 7th Working IEEE/IFIP Conf. on Software Architecture, Vancouver, Canada, pp. 231-240.

Neuliep, J. W., 1991. *Replication Research in the Social Science.,* SAGE publishing.

Nuseibeh, B., 2001. Weaving Together Requirements and Architectures. IEEE Comp., 34(3): 115-117.

Nuseibeh, B.; Easterbrook, S., 2000. Requirements engineering: a roadmap. Proceedings of the Conference on the Future of Software Engineering. ACM Press, pp. 35-46.

Ramesh, B. and Jarke, M., 2001. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, Volume 2, Issue 1, pp. 58-93, January 2001.

Rapanotti, L., Hall, G., Jackson, M., Nuseibeh, B, 2004. Architecture-driven problem decomposition. Proceedings of the 12th IEEE International Requirements Engineering Conference, Kyoto, Japan, pp. 80–89.

1st International Workshop on Replication in Empirical SE Research (RESER), Cape Town, South Africa, May 2010.

Runeson, P, and Host, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Journal of Empirical Software Engineering, Vol. 14, #2, pp. 131-164.

Wieringa, R.J., Heerkens, J., 2006. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. RE Journal 11, pp. 295–307.

Chapter 7

Non-technical factors' impact on architecting[47]

1 *Introduction*

It was more than twenty years ago that Curtis et al. (Curtis et al., 1988) identified the importance of human-based non-technical factors (such as communication skills, leadership skills, and professionalism) in design processes for the success of large projects. Despite the importance of such factors, in recent Software Engineering (SE) educational research, sentiments have been expressed that SE graduates are not well prepared for their professional careers due to their lack of non-technical skills training (Karunasekera and Bedse, 2007; Taran, 2008).

If there is one area of system development, more so than most others if not all, where SE personnel need to have non-technical factors under control then it is "Systems Architecture". This is because an architecture is not only a technical artefact of the system, it is a *key* artefact that is of interest to many different types of stakeholders (Bass et al., 2003), requiring leadership and other qualities in an architect so as to manage stakeholder relationships and expectations (Bredemeyer, 2006; Clements et al., 2007). For example, from the end users' perspective on operational quality, the customers' perspective on what they are buying or how much the system will support their business processes, the management's perspective on system implementability and cost, the tester's perspective on test plan prioritisation and scheduling, and from an integrator's perspective on which components and how they will be integrated to yield the desired system, etc., the system architect needs to be multi-faceted.

Our own discussions with senior members of large organizations suggest that

[47] This chapter is an extended version of (Ferrari et al., 2009).

recent university-level graduates, while being well-trained in technical issues, often lack the non-technical skills for entry into industry to be effective junior architects. Such concerns are not surprising considering that most of the research and pedagogical literature about SA are on technical issues (such as design methods, architectural notations, patterns and styles, analysis, etc. [Bass et al., 2003]), and only little on human-centred factors.

Recently, however, there has been increasing interest on the importance of non-technical factors in Systems Architecture (Bass and Berenbach, 2008; Clements et al., 2007a). Some of these non-technical factors include: leadership, communication, inter-personal, project management, work habits, etc. (Clements et al., 2007b). Research that has been done includes building a model of architect competency (Clements et al., 2007b), design of architecture skills assessment instrument (Downey and Babar, 2008), and observations of the human-factors effect in security risk management (Islam and Creighton, 2008).

Despite this recent research, there has been no empirical study conducted on the preparedness of university graduates for a career as system architects in industry. In particular, the concerns voiced by others on the lack of non-technical skills amongst the graduates (Karunasekera and Bedse, 2007; Taran, 2008) prompted us to investigate the following research question from an educational perspective: *What is the impact of non-technical factors on Systems Architecture?*

To answer this question, we conducted an empirical study on 15 student architecting teams developing an architecture from the same set of requirements. Specifically, we identified the type of non-technical based problems that students have, and provided a quantitative breakdown of these problems. Examples of such problems include *procrastination*, *poor planning*, and *missing/late for meetings.* Additionally, we examined how these problems affected the product quality of the teams' architectures. The findings from this study are meant to provide much-needed empirical evidence in this area.

Furthermore, to identify the possible causes of the weaknesses identified in student/junior architects, we analysed the current state of the IEEE/ACM software engineering (SE) (IEEE/ACM, 2004) and computer science (CS) curriculums

(IEEE/ACM, 2005). The findings from this assessment were then analysed together with the findings from the student projects to determine whether or not there is any correspondence between the two.

Our general finding is that non-technical issues are under-represented in the current curriculums. For example, only 7% of the total hours in a recommended curriculum are allocated to non-technical factors. Based on this analysis, the main contribution of this paper is then a set of recommendations for improving education of students who are interested in becoming software architects in industry. These recommendations centre on suggestions for improvement to the IEEE SE and CS curriculums, and for SE educators to be active in recommending to SE/CS students relevant non-technical focused courses outside of SE.

The rest of the paper is organized as follows. In the next section we describe related work; in section 3 we discuss the empirical study on student architecting projects; in Section 4 we present analysis of the current ACM SE and CS curriculums; Section 5 discusses correspondence between the results from sections 3 and 4; in Section 6 we give recommendations for improving architect education in academia; and lastly, Section 7 concludes the paper.

2 Related Work

In this subsection, we overview related work pertaining to research done on non-technical factors in SA, and also research focused on improving SA education.

In (Clements et al., 2007b), the authors report on a survey that was conducted to identify the prominent duties, skills and knowledge of a software architecture. Their analysis is focused on three key categories for architecting success: duties, skills and knowledge. Within the area of architecting duties, their analysis shows that the specific duties of Project management, requirements, architecture evaluation, analysis, and interaction with clients were the most prominent. Under the skills category, the survey found that the prominent sub-categories were communication, inter-personal skills (within team), leadership, and the ability to handle the unknown. Lastly, under knowledge, computer science knowledge was considered the most valuable type of knowledge, followed by architecting conceptual knowledge, platforms and technology knowledge, and organizational knowledge. The output of this work is a model of architect competency that can be used as a basis for hiring and training processes, and curriculum development.

(Berenbach, 2008) expands on the previously identified list of skills and knowledge, and the discussion is in the context of observations from industry. Examples of the identified skills are the understanding of the impact of testing and maintenance needs, ability to understand and resolve stakeholder conflicts, and the ability to efficiently train project team members. In addition to discussing architecting skills, Berenbach identifies key personality traits for successful architects. These include: attention to detail, ability to listen, ability to motivate and coach, and having an "open mind". Furthermore, he discusses the gap between the perceptions of what non-technical attributes competent architects require in academia and industry. Our current paper complements the work described in (Berenbach, 2008); in particular, the empirical study in our paper is meant to provide concrete findings that can support the more intuitive-based discussion provided in (Berenbach, 2008).

A previous study of ours (Ferrari and Madhavji, 2008) reports on an empirical study examining one type of knowledge (Requirements Engineering [RE]) that could have an impact on SA. Specifically, we conducted a controlled-study that investigates the impact of software architects having RE knowledge and experience when performing SA. Two types of study groups were used where: one type of group had previous training and/or experience in RE; whereas, the other type of group did not. Both types of groups conducted the same architecting project given the same initial set of requirements from the banking domain. The results show that the architects with RE knowledge/experience produced a significantly better architecture (10% difference in the overall architectural quality as measured through some 12 architecting areas). The study also highlighted specific architecting areas where these architects performed better. Examples of these areas include: determining architectural tactics, selecting/creating an architectural pattern to satisfy key quality drivers, and interface specification.

In (Downey and Babar, 2008), Downey and Babar present an instrument to collect data for the empirical assessment of the needed skills for a software architect. Unlike in (Clements et al., 2007b), which was based on a survey of many different sources (not necessarily architecting practitioners), the work in (Downey and Babar, 2008) is focused on collecting data directly from practitioners from a socio-cognitive perspective. The authors' ongoing work is to use this instrument to interview architects in industry to create an empirical based profile of architects skills and competencies.

The issue of student architects not being properly trained through university education was raised at a working session at WICSA 2005 (Shaw and Van Vliet, 2005). The overall goal of the session was to identify shortcomings in SA courses at university-level institutions. Specifically, issues that were identified and discussed were: how to make SA courses sufficiently realistic? How to teach non-technical competencies? How such an SA course can fit within current curricula? And, what further training to software architects need beyond the university setting? Further to the identification of the above issues, the participants of the session created a map of the various architecting competencies (e.g., people skills, architecting techniques, domain knowledge, etc.) and their relative importance for different types of students (CS graduates, SE graduates, SE graduates with specialized SA training).

Following this working session, as reported in (Mannisto et al., 2008), Mannisto et al. describe the design and evaluation of an advanced course for teaching systems architecture in academia which considers the issues raised in (Shaw and Van Vliet, 2005). The goals of the course are to explicitly train and raise awareness for student architects on issues that are currently not taught in SA courses but are relevant for industrial practice. These issues include: architecting in the context of an unclear problem definition, knowing when the solution is "good enough" to begin detailed design and coding, and dealing with a priori constraints from architecting in an evolutionary context.

While all the described efforts seem worthwhile in making progress in the field of system architectures, there is a clear research gap in the community's understanding of the kinds of problems, especially non-technical ones that exist in student architecting projects. A deeper understanding of this would help in improving SA curricula and in ensuring that students are better prepared for architects' positions in industry software projects. This paper is a step in the direction of filling this gap.

3 Empirical Study on Software Architecting Projects

In this section, we describe the empirical study on software architecting projects. This includes an overview of the study (variables of interest, participant description, data collection and analysis, threats to validity), and the results of the analysis.

3.1 Study Overview

The empirical study, conducted in the university setting, investigated the following research question: *what is the impact of non-technical factors on SA?* The main variable of interest from this question is the *non-technical factors,* and the corresponding metrics are the type and frequency with which they occurred during the architecting process. As discussed in the Introduction section, such factors include: leadership, communications, inter-personal, project management, work habits, etc.

The study involved senior-level university student architects working on a two-month architecting project in an SA course. In total, there were 60 participants with 4 participants allocated to each team, giving a total of 15 architecting teams.

Each team had to design an architecture given the same set of requirements. The requirements were created externally to the project, and in total there were approximately 80 high-level requirements. Given these requirements, each of the 15 teams independently developed an architecture using the ADD method (Bass et al., 2003). Examples of key steps in ADD include: choosing architectural drivers from requirements, determining architectural tactics to satisfy drivers, and identifying child modules during decomposition to implement the tactics.

For identifying non-technical factors, there were two sources of data: audio recordings of student/staff help-sessions and intra-team e-mail communications (approximately 80 e-mails average per group). It was mandatory that each team met with the course staff once a week minimum to discuss their project; this resulted in over 100 hours of audio that was later transcribed. It was not possible to observe intra-team's working-sessions to obtain live data.

Content analysis (De Vaus, 2002) was used to analyze the transcribed textual data and e-mail communications. In short, the analysis technique works by scanning through the text and categorizing text segments of interest. For example, text fragments that loaded significantly on variables such as "Missing/late for meetings", "Delivery of inadequate work", "Procrastination", etc. would be tagged with the corresponding category-labels. These can then be counted to create frequency figures of the various categories.

The other source of data used in the analysis was the assessment of quality of the teams' architectures, which was used to corroborate findings with the frequency of non-technical factors encountered in the transcribed data and emails. The quality-criteria for the instrument items were derived from SA literature, and the standard templates from (Bass et al., 2003). The central components measured by the instrument include: Model the environment; Use Cases; Quality Scenarios; Module Decomposition structure; Component and Connector structure; Deployment structure; Interface specification; Modelling the dynamic behaviour of the system; Overall Architectural properties; Architectural reasoning; View descriptions; and Overall documentation quality. This instrument was used by five experienced software engineers (with experience ranging from 5 to 27 years in SE and research) to assess the resultant architectures from the study projects. Also, to ensure content and face validity (Johnson and Christensan, 2004) of the instrument, there were numerous iterations and stages in the design and implementation of the instrument. This included reviews and establishing relative weights for di□erent items corresponding to the project requirements, and had intimately involved six knowledgeable software engineers with RE and SA experience. The instrument is described in more detail in (Ferrari and Madhavji, 2008).

3.2 Threats to Validity

Based on (Johnson and Christensan, 2004), four types of threats that might apply to the multi-case study described here were identified: External, Construct and Conclusion validities, and reliability. Because we are not attempting to demonstrate causality between variables, threats to *internal* validity are not a concern.

3.2.1 External Validity

External validity refers to the degree to which the results of a study can be generalized across a population, setting, and time (Johnson and Christensan, 2004). Threats to external validity occur when researchers draw incorrect conclusions based on the sample data (Johnson and Christensan, 2004). We discuss only the threats to *population validity* as other types of threats did not apply.

Population validity – this threat refers to the generalization of the sample to the population, and the sample results to the different types of people within the population. In our study, this threat exists since our study was conducted in a single university setting and it is possible that the overall student's maturity and proficiency could differ among universities, leading to potentially different results. However, the students educational backgrounds at the university level will be similar since, in general, educational institutions incorporate the IEEE SE and CS curriculums within their degree programs (see Section 4). Further to this, our discussions with senior-level members of a large-scale organization suggest that students coming into industry, no matter from which university, encounter non-technically oriented difficulties in industry.

Also, our study focused on fourth-year undergraduate students who may perform differently than graduate-level students; therefore there is a threat of generalizing these results to graduate students.

3.2.2 Construct Validity

Construct validity is the degree to which inferences can be made from the measures in the study to the theoretical constructs on which those measures were based. In our study, the variable of interest *non-technical problems* was measured by collecting data directly from the study participants (through interviews and e-mail communications). To mitigate the threat of *content* and *face* validity, the data analysis procedure described in Section 3.1 was validated through piloting, independent review and re-coding. Furthermore, the various categories of non-technical problems were identified *a priori* based on discussions with senior-level members of a large organization and existing literature. This *apriori* identification is important since the data gathered is readily related to the constructs of interest (i.e., *non-technical problems*).

3.2.3 Conclusion Validity

Conclusion validity is the degree to which conclusions we make based on our findings are reasonable (Johnson and Christensan, 2004). There are three ways in which threats to conclusion validity can be mitigated in a quantitative-based study: statistical power, reliability, and proper implementation of study methods. In our study, statistical power is mitigated in our co-relational analysis through the use of 15 data points (i.e., the fifteen teams). As discussed in section 3.2.2, we used multiple researchers to review the coding of the data, to alleviate potential *researcher bias* increasing reliability of the data. Furthermore, *data triangulation* was used through the investigation of different sources of data (meeting and e-mail communications), again to promote data reliability. Lastly, the researchers performing the coding were trained prior to the actual codings to ensure they properly carried out their task.

3.3 Results

In this sub-section, we present and discuss the results of analyzing the non-technical problems in the student projects. In total, we identified 156 problems, spread out amongst the 15 teams and across various identified categories. This overall result is conservative because we only had access to staff/student help sessions and e-mail communications (see Section 3.1). Other sources of data such as observations of a team's project meetings or intra-team communications (such as instant messages) would have provided additional insight into these non-technical problems. In view of this conservative measurement, the 156 problems we noted can be considered substantial.

Apart from the type and frequency of non-technical problems encountered, it is also important to assess the impact of these problems on the end result – the resultant systems architectures. The second sub-section deals with this aspect.

3.3.1 Profile of Non-technical Problems

Figure 7-1 shows the details of the quantitative breakdown of the non-technical problems encountered in the architecting process in the various categories. Below, we discuss these findings.

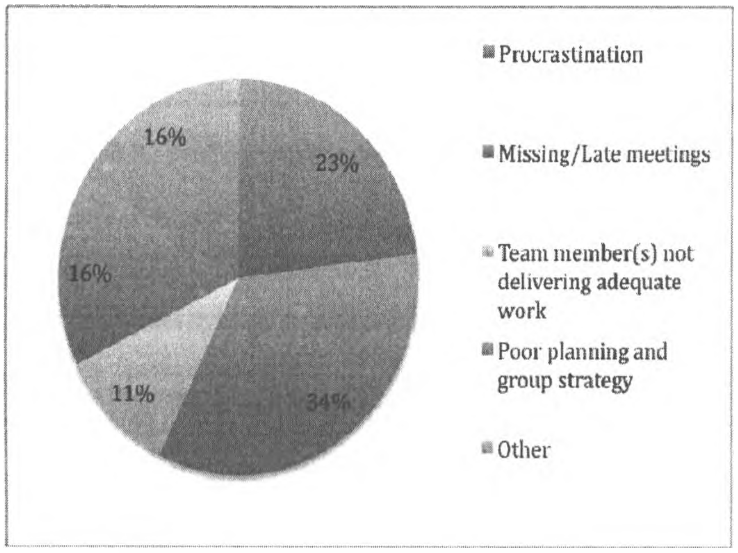


Figure 7-1. Profile of non-technical problems

<u>Missing/late for meetings (32%):</u> This is when group members either missed or were late for meetings and did not provide sufficient notice as to their tardiness. This category constituted the largest percentage of problems that occurred in the teams' process. This obviously led to many specific problems in the process, such as project delays, not having all group members "on the same page" in terms of what deliverables were expected, and a lack of understanding of emerging product details. The following is a quote from a student describing an instance of this problem and the consequence:

"The problem that I'm having right now is that the work I'm doing should be in accordance to what is happening at the system interface level, which has been left to *anonymous* (a team member). However, he has missed the last couple meetings so I'm not exactly sure why he made a few decisions the way he did, so it's difficult for me to proceed."

Procrastination (22%): This is when either individuals or the entire group pushed back expected work until a later due date, or left work items to be completed until just before an expected due date. This result is not entirely surprising; in (Berenbach, 2008), it is observed that this is a common problem seen in students and the work that they perform. The root of the problem lies in the fact that the time frame for a student project is often much shorter than an industrial project, so it is possible to delay the work and still complete the project, and also that the final result is to "get a passing grade"; whereas, in industry, real customers must be satisfied with the end result, leading to harsher consequences for unsatisfactory work. The following student quote highlights this issue and the possible cause:

"Well I think maybe what's going on is we're taking other classes and working entirely on those assignments, since they're due sooner, and leaving this one to the very end, which isn't a good thing but this is the way we've always worked. And now, we don't have much to show for this project and little time left to complete -- what threw us off was the way the project was structured and nature of architecting work, it's a lot different than assignments/projects in our other classes."

Poor planning and group strategy (15%): This involved the team's planning of deliverables, meetings (times, setting, agenda) and overall group strategy (e.g., work more individually and combine results, or perform the majority of the work as a team). This category was impacted mostly by the ability of the designated lead architect to organize the project. Some of the more common problems within this category were: meeting agendas were not planned ahead of time, so some individuals of teams did not arrive prepared; contingency plans were not in place for unforeseen events (e.g., group member becomes sick and cannot deliver expected product); and tasks and their dependencies were not well thought out, often, tasks were assigned to individuals without analyzing or discussing who would be best suited to a particular task. The following quote describes an instance of this problem relating to decision-making strategy of the group:

"The one thing that I was really having problems with was everyone going their own way doing their work and coming back with four really great diagrams that don't really agree with each other, I know we originally thought that everyone could work on the same thing independently to have more ideas but this strategy has ended up wasting a lot of time and effort."

Individuals delivering inadequate work (14%): This problem category is where an individual in the team was given a task and a deadline, and when they submitted the work it was incomplete, late, or lacking in details. This had a direct impact on the number of iterations required to complete the project, as rework was required. Also, other team members' work suffered when they were expecting completed work from their co-workers. Although this problem seems to be rooted in an individual's lack of work ethic or intelligence, there were several cases where the cause could be traced to the deficiency in the expression of the project or architectural plans, in particular, by the lead architect. The following is a quote from a student denoting an instance of this problem, "here you go guys, I could not quite finish this sub-system as expected. When I went to do the work I didn't really understand what we had brainstormed the other day and we don't have a soft-copy so this is all I could do."

Other (16%): Other categories such as *lack of leadership, communication issues,* and *mistrust between team members* each had several problems, however, these were not as high as the aforementioned categories.

3.3.2 Non-technical Factors and Architectural Quality

Further to the identification of the most common non-technical problems as identified in the previous subsection, we examined the relationship between these problems and the final architectural quality. The correlation between the frequency of problems a team had and final architectural quality was -0.51, with a statistical significance of p=.02, meaning this result was most likely not due to chance. The -0.51 co-relation means that as the frequency of non-technical problems increases, the architectural quality decreased. The strength of this relationship is classified as "medium".

This is not entirely surprising, as in our previously published work (Ferrari and Madhavji, 2008) we found that an architect's past academic performance and whether an architect had Requirements Engineering knowledge and experience were the most significant factors for determining a team's success in architecting. Regardless, examining more closely the team's data, we do find some interesting points.

Team #	# of Non-technical problems	Architectural Quality (out of 100)
1	4	100
2	4	88
3	11	82
4	21	73
5	9	61
6	16	77
7	12	60
8	7	65
9	13	45
10	12	48
11	6	83
12	15	62
13	6	83
14	15	41
15	5	72

Table 7-1. Team breakdown of non-technical problems and architectural quality

Referring to Table 7-1, out of five (of 15) teams that had the least number of non-technical problems, four of these were the highest performing teams in terms of final architectural quality. Conversely, three of the five teams that had the most number of non-technical problems had the weakest architectures.

Also, the four teams with the lowest quality architectures were in the top five for the most *procrastination* problems (3, 3, 4 and 4 instances of the problem for each of these three teams respectively). When further examining the details of the architectural quality of these three teams, we see that the scores most affected by the *procrastination* issues were their *deployment view(s)* (received a 0 out of 100 – these teams did not submit any work in this area suggesting that for their particular process they were leaving this more to the end and did not have time to work on it), *architectural reasoning* (average of 40.5), *detailed interface specification* (average of 54), and *behavioural modeling* (average of 30). The latter two areas are typically done more at the back-end of the architecting process so the *procrastination* issue clearly led to them not having enough time to do a proper job in these areas and the quality suffered. *Architectural reasoning* deals with the quality of the given rationale, assumptions, alternative design decisions, and also of any preliminary analysis of the architecture. For these teams, they simply submitted a first draft of this work and did not have time to fully flesh out the details.

Other than *procrastination,* none of the other specific non-technical problem categories (e.g., *missing/late for meetings, poor group planning/strategy,* etc.) could be clearly corroborated with the overall architectural quality scores; however, this does not mean that specific instances (not necessarily the total frequency) did not have an impact on specific technical architecting areas. We are currently performing further analysis to link specific instances of non-technical problems to specific problems in the architecture.

Nevertheless, the findings presented in this sub-section do suggest that the non-technical problems (or lack of) had an impact on the SA quality, but further studies would need to be conducted to strengthen this claim.

4 Analysis of the IEEE/ACM SE Curriculum

The purpose of the IEEE/ACM SE curriculum (IEEE/ACM, 2004) is to provide guidance and recommendations to academic institutions regarding the design of SE undergraduate degrees. In this curriculum, it is recommended that an SE degree should consist of a total of 475 contact hours, with approximately 3 hours of outside lecture time spent per lecture hour. The curriculum allocated this total time into 10 broad categories, examples of which are: *computing essentials, mathematical and engineering fundamentals, software modeling and analysis,* etc. Nine out of the ten categories are heavily technically oriented. The one category that deals with non-technical factors is *Professional practice.* Specifically, this category contains the following topics: *group dynamics/psychology*, *communication skills* (specific to SE), and *professionalism.* This category is recommended an allocation of 35 hours (approximately 7% of the total) to be sub-divided into the aforementioned topics.

Based on the specific items listed within each of the topics (for example, under communication skills there is *Reading, understanding and summarizing reading (e.g., source code, documentation),* most of the hours in the sub-categories would be subsumed within technical projects (for example group projects). Thus, the non-technical learning in software projects occurs through "implicit" learning; students would not explicitly learn foundational theory, principles, guidelines, etc. concerning non-technical factors.

This corroborates with our own observations of students within various academic institutions; for example, often when a student, even at the doctoral level, gives a presentation based on their research they flounder on such issues as an appropriate title, succinctly describing their research in a few slides, and convincing the audience of the cost-benefit of their research. They are expected to give presentations, but are never actually taught how to effectively create and deliver one.

Software engineers in practice will not necessarily have graduated with a SE degree, but instead may have a Computer Science (CS) degree and background. The reason for this is that many universities do not offer SE-only degrees, but instead will offer CS degrees with electives in, or specialization in, SE. When examining the ACM CS (IEEE/ACM, 2005) curriculum with respect to learning about non-technical factors, the situation is even worse than in SE. While there is a set of knowledge areas devoted to non-computing topics (i.e., non-technical factors) such as organizational theory, decision theory, and organizational behaviour, the weights assigned for most areas on a scale of 0 to 5 (where 0 means not important in CS) was 0. The only areas that were assigned a weight greater than 0 were *interpersonal communication* and *project management.*

The only other option to formally learn non-technical factors is through a student's non-CS and non-SE elective courses. Such electives enable students to take courses that are more aligned with non-technical factors such as psychology, business, and management. However, based on our own observation of university curriculums, the number of electives a student can take is quite limited (typically around 1 full course per year) and there are also no recommendations or guidelines from within an SE or CS degree on which, or how many, courses outside of the CS/SE degree would be useful and for what type of purposes. Therefore, it is not surprising that students lack substantial experience in non-technical factors when involved in group projects.

5 Discussion

In this section, we discuss the correspondence between the findings from the student projects in section 3, and the analysis of the ACM SE curriculum in section 4.

Recall that in Section 3 we presented and discussed results of a student architecting project where the teams encountered 156 problems rooted in non-technical factors (*missing/late for meetings, procrastination, poor planning, etc.).* These problems, in many cases, led to reduced architectural quality in their final deliverable. Based on our discussions with senior members of large organizations, these types of problems are highly problematic in industry and do frequently occur. Despite the severity of the consequences of these problems in the real-world, in academia it seems that many students, even by the time they are nearing completion of their degrees, still have not learnt the importance of "real-world" skills. These findings corroborate with observations made by others (Berenbach, 2008) -- of a gap between the attitudes in industry to academia.

To reason about the cause of this gap, we analysed the ACM SE and CS curriculums (IEEE/ACM, 2004/2005) in order to discern how much emphasis was placed on the formal education of non-technical factors. Our findings suggest that these non-technical skills are seriously under-represented. Specifically, the weaknesses identified in the current curriculum "do" seem to correspond to the problems encountered in the student projects; the lack of hours devoted to such issues as *communication* and *professionalism* will have a direct impact on a student's performance in these areas.

Furthermore, specific types of problems as identified by our study, and the implications of committing such problems, are not accounted for in any way in the curriculum. For example, the specifics of *professionalism* present the broad topics of ethics, legal issues, organizational behaviour, etc., but these are not discussed in a detailed way so that educators and students have a firm understanding of the "What" and the "How" of these broad categories. Furthermore, the fact that only twenty hours of teaching time are recommended over a span of a four-year degree, we believe, severely limits the ability to adequately educate/train students in these areas.

Despite the apparent correspondence between the weaknesses identified in the student projects and the SE and CS curriculum, controlled studies would need to be conducted to determine causality. In such a study, the dependent variable would be the quality of the architecting process and quality of architecture, and the independent variable would be whether a participant had formal education/training in non-technical factors. This design of the study, however, would be difficult to conduct because of the problem in finding participants to form the control/non-control groups.

6 Recommendations

Based on our findings, we have two recommendations that could improve the problem of students not being educated/trained in non-technical factors. First is that CS and SE departments in universities should explicitly provide guidance and information to students on which courses they can take outside of the department (such as organizational psychology, management and business) that would be useful for excelling at positions in industry that require more than basic technical skills. The implementation of such guidance and information delivery would be university or department specific, because each academic institution has its own curriculum and departmental structure.

Second, that the SE and CS ACM curriculums be revisited to elaborate on the non-technical skills sections. Senior-level members of large organizations have stressed that it is easier to find people with good technical skills than finding people that excel in both technical and non-technical skills (Karunasekera and Bedse, 2007). The first suggestion is to simply increase the weight of non-technical education in SE and CS degrees. The second curriculum improvement can be to tailor educational suggestions for different career paths in SE. For example, educational curriculum recommendations can be different for a student aiming to be a software architect (which would have more emphasis on non-technical skills) vs. a student aiming at a career in software testing (where skills-base should be more technical).

7 Conclusions and Future Work

Systems Architecture has traditionally been seen as a mostly technically based discipline (Bass et al., 2003). However, recently there has been research in identifying and understanding the role that non-technical aspects have in architecting (Bass and Berenbach, 2008; Clements et al., 2007a). We continue this line of research by investigating the impact of human-based non-technical factors in Systems Architecture. To investigate this issue, we conducted an empirical study in a university setting to determine what type of non-technically based problems architecting students had, and the impact of these problems.

Based on the findings of our study (see section 3), we conclude that senior-level students do encounter many problems rooted in non-technical factors. The 15 architecting teams had 156 non-technical problems, the majority of these being in the areas of: *missing/late for meetings (34%), procrastination (23%), poor planning and group strategy (16%),* and *individuals delivering inadequate work (11%).*

Additionally, we analysed the ACM SE and CS curriculums (IEEE/ACM, 2004/2005) in order to determine how much emphasis was placed on the formal education of non-technical skills (see section 4). Our findings suggest that these non-technical skills are under-represented. We make specific recommendations for SE-

CS curriculum improvement (see section 6) that would aid in the training and education of software architects.

For future work, further empirical research is needed in this area. As suggested in Section 5, controlled studies would be beneficial for establishing causality between non-technical factors and the success of an architecting project. Case studies could also be conducted examining the effect on SA of issues such as compatibility of the team-members' personality, team's heterogeneous skill sets, and, team politics and trust.

References

Bass, L., Berenbach, B., 2008. Leadership and management in software architecture (LMSA'08): a report on an ICSE workshop. ACM SIGSOFT Software Engineering Notes 33(4): 27-29.

Bass, L., Clements, P., Kazman, R., 2003. *Software Architecture in Practice*. Addison-Wesley.

Berenbach, B., 2008. "The Other Skills of the Software Architect", Leadership and Management in software architecture (LMSA'08), Leipzig, Germany, pp. 7-11.

Bredemeyer, D., Malan, R., 2006. The role of the architect. Architecture Resources for Enterprise Advantage. Bredemeyer Consulting. Online link: http://www.bredemeyer.com/, last accessed April 2009.

Clements, P., Kazman, R., Klein, M., 2007. Working Session: Software Architecture Competence. WICSA 2007.

Clements, P. C., Rick Kazman, Mark Klein, Divya Devesh, Shivani Reddy, Prageti Verma, 2007. The Duties, Skills, and Knowledge of Software Architects. WICSA 2007.

Computing Curricula for Undegraduate Degree Programs in Computing Discipines. Volume of the Computing Curricula Series, September 2005, The Joint Task Force on Computing Curricula, IEEE Computer Society, ACM.

Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Volume of the Computing Curricula Series, August 23, 2004, The Joint Task Force on Computing Curricula, IEEE Computer Society, ACM.

Curtis, B., Krasner, H., Iscoe, N., 1988. A Field Study of the Software Design Process for Large System. Comms of the ACM 31(11), 1268–1287.

De Vaus, D.A., 2002. *Analyzing Social Science Data*. SAGE Publishing Ltd, London.

Downey, J., and Babar, M. A., 2008. "On Identifying the Skills Needed for Software Architects", Leadership and Management in software architecture (LMSA'08), ICSE workshop, Leipzig, Germany, pp. 1-5.

Ferrari, R., and Madhavji, N. H., 2008. Software architecting without requirements knowledge and experience: What are the repercussions?. Journal of Systems and Software, Volume 81, Issue 9 (Sept. 2008), Pages 1470-1490.

Ferrari, R., Wilding, M., Madhavji, N. H., 2009. The Impact of Non-Technical Factors on Software Architecture. 2nd Workshop on Leadership and Management in Software Architecture, Vancouver, Canada.

Islam, S, and Creighton, O, 2008. Human Factors in Software Security Risk Management. Leadership and Management in software architecture (LMSA'08), Leipzig, Germany, pp. 13-16.

Johnson, R. B., and Christensan, L., 2004. Educational Research: Quantitative, Qualitative and Mixed Approaches. Allyn & Bacon; 2nd edition.

Karunasekera, S., and Bedse, K., 2007. Preparing Software Engineering Graduates for an Industry Career. 20th IEEE/CS Conference on Software Engineering Education and Training (CSEE&T 2007), Dublin, Ireland, pp. 197-206.

Mannisto, T., Savolainen, J., Myllarniemi, V., 2008. Teaching Software Architecture Design. Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), Vancouver, Canada, pp. 117-124.

Shaw, M., and van Vliet, H., 2005. Software Architecture Education Session Report. Fifth Working International Conference on Software Architecture, 2005, Pittsburgh, US, pp. 185-190.

Taran, G., 2008. Managing Technical People: Creatively Teaching the Soft Skills of Human Interaction in Today's Diverse Classroom. 21st IEEE/CS Conference on Software Engineering Education and Training (CSEE&T 2008), Charleston, US, pp. 93-100.

Chapter 8

Emerging Theory

1 *Introduction*

The Requirements Engineering (RE) and Systems Architecting (SA) discipline is abundant in technology, especially for aiding in the transitioning from RE to SA, along with its accompanying underlying *construction (*or *engineering)* theory (STRAW, 2001 and 2003). However, "empirically grounded theory" on the interaction between RE and SA in terms of the *human* and *process* factors involved in RE and SA development is scattered and mostly anecdotal. Furthermore, there is a near total absence of any grounded theory (*construction*, *human involvement*, *process*, etc.) on conducting RE in the presence of an existing SA (Jackson, 1994; Shekaran, 1994) - this issue is orthogonal to mainstream RE and SA development and how RE and SA is taught in pedagogical literature.

Thus, an emerging descriptive theory is proposed (see later in Section 3) that describes the impact of *human* and *process* factors in the interaction between the RE and SA processes. The emerging theory is developed "bottom-up" based on the evidence from the six empirical studies described in Chapters 2-7 of this thesis. The theory building followed the *hypothetico-inductive* model (Sjoberg et al., 2008), which means that the theory is inferred directly from observational data.

There are numerous implications of the proposed emerging theory for RE and SA practice and research which are also described in this chapter.

The chapter is structured as follows. In the next subsection, an overview of the six studies from which the emerging theory is derived is given; Subsection 3 describes the emerging theory; In Subsection 4, the emerging theory is evaluated based on the guidelines from (Boehm and Jain, 2006) and (Sjoberg et al., 2008). Lastly, Subsection 5 concludes the chapter.

2 Overview of studies

Before describing the emerging theory, a brief description of the six studies from this thesis is given. Table 8-1 provides this brief summary information and consists of a study ID, main research topic, empirical approach, context and reference to the chapter in this thesis where more information can be found.

Study ID	Main topic	Empirical approach	Context	Reference
S1	Impact of RE knowledge on architecting	Controlled experiment	Architecting course project	Chapter 2
S2	Requirements-oriented problems while architecting	Multiple case study	Architecting course project	Chapter 3
S3	Characteristics of new requirements in presence/absence of existing architecture	Controlled experiment	Requirements course project	Chapter 4
S4	Impact of SA on requirements decision-making	Multiple case study	Requirements course project	Chapter 5
S5	Impact of SA on requirements decision-making (replication and extension of S4)	Case study	Large-scale innovative rail project	Chapter 6
S6	Impact of non-technical factors on RE and SA	Multiple case study	Architecting course project	Chapter 7

Table 8-1. Summary of studies

3 The Emerging Theory

Before describing the emerging theory in detail, we first describe the necessary background information on how the theory is organized and presented.

In (Sjøberg et al., 2008), three levels of abstraction for theoretical propositions are identified, where Level 1 propositions are minor working relationships that are concrete and based directly on observations, Level 2 are theories of the middle range that involve abstraction of possibly many Level 1 theories but are still closely linked to observations, and Level 3, all-embracing theories that seek to explain an aspect of Software Engineering.

The preliminary theory propositions are given in Table 8-2 and Table 8-3, for human and process factors respectively, both of which are described in more detail in the next two subsections. The tables are hierarchically structured to reflect the different levels of abstraction denoted in the propositions. The propositions at Level 1 are mostly directly based on emergent hypotheses and implications of results from the six studies in this thesis (described in Chapters 2-7); whereas, Level 2 propositions comprise higher-level abstractions of the Level 1 propositions. Essentially the Level 1 propositions are more concrete and testable sub-issues that all serve to test the more abstract Level 2 propositions. Because of the emergent nature of the theory and the exploratory nature of the six studies, the given propositions are mostly at Level 1, with six at Level 2. There are no Level 3 theories given; these are typically derived from a much larger set of related study findings when the discipline is more mature (Sjøberg et al., 2008). Table 8-2 and Table 8-3 also contain a reference to the study where the Level 1 proposition is derived (in Column 3), and key summary terms (in Column 4).

For each theory theme (human or process factors) and the Level 2 statements, the grounded evidence used to form the theory will be presented. This is equivalent to logical or mathematically based proofs being given for a more prescriptive or mathematical theory. Each proposition has a unique ID that is used to reference specific propositions in the Tables in the descriptive text accompanying the Tables.

3.1 Human factors theory

Level 2	**Level 1**	**Source**	**Key terms**
P1. Different types of RE and SA knowledge possessed by the human agents conducting RE and SA processes have a significant effect on RE and SA products and process quality.	P1.1 Software architects with education and training in RE positively impact resultant architecture and architecting activities.	S1, S2	Knowledge factors
	P1.2 Requirements analysts with education and training in SA positively impact resultant requirements and requirements activities.	Logical extension from S1	Knowledge factors

P2. Varying types of skill-sets and personal interests (such as more technologically motivated vs. user-needs motivated) possessed by the human agents conducting RE and SA processes significantly alter resultant RE and SA product characteristics.	P2.1 A requirements elicitation team with motivation and expertise in system solution is more likely to elicit requirements that have technological bias.	S3	Personal interests factors
	P2.2 A requirements elicitation team with motivation and expertise in a system's context (e.g., human-computer interaction and end-user satisfaction) is more likely to elicit requirements that are user-focused.	S3	Personal interests factors
P3. Human factors such as mental capability, education, and others (e.g., professionalism, communication, leadership, etc.) significantly override the impact of technological usage in an RE and SA project.	P3.1 Non-technical factors training and education reduces non-technically oriented problems in RE and SA.	S6	Non-technical factors
	P3.2 Mental capability, education and experience are the highest determinant factors for predicting RE and SA product quality.	S1, S2, S5	Mental capability factors
	P3.3 The employment of RE and SA technology does not significantly decrease variance between project outcomes in terms of RE and SA product and process quality.	S1, S3, S5, S6	Technolog y factors

Table 8-2. Human factors theory propositions

1. *Different types of RE and SA knowledge possessed by the human agents conducting RE and SA processes have a significant effect on RE and SA products and process quality.*

In S1, it was found that a strong contributing factor in the quality of the final architecting products was the type of education possessed by the architects; specifically those with RE education performed better than those without (approximately 10% difference between the two types of groups – See Chapter 2 Section 4.2). Also in S1, it was qualitatively determined technical areas (such as *determining architectural tactics*, *modeling quality scenarios* and *pattern determination*) where feedback was required during the architecting process to overcome project difficulties – again teams with RE knowledge performed better (see Chapter 2 Section 4.2). This phenomena was also observed in S2, architects with RE knowledge had less requirements-oriented problems in-process than those without RE knowledge (see Chapter 3 Section 3.2). These findings led to proposition P1.1.

One knowledge factor that was not explicitly studied, due to contextual constraints, is the impact of architecting knowledge on RE. Therefore, proposition P1.2 was derived and is based on the logical assumption that critical knowledge from SA would be useful for, at the very least, parts of the process and product quality of RE.

2. Varying types of skill-sets and personal interests (such as more technologically motivated vs. user-needs motivated) possessed by the human agents conducting RE and SA processes significantly alters resultant RE and SA product characteristics.

This proposition is entirely inferred from the observations in S3, where one of the side-results of this study is that human factors such as personal interest and motivation influenced specific requirements characteristics (degree of tech focus vs. user-focus, etc.), but not necessarily the "goodness" of the requirements, leading to P2.1 and P2.2. These attributes, coupled with presence/absence of technical artefacts (such as the existing SA) 'boosted' specific product characteristics (e.g., analyzing existing SA output requirements that were tech-focused, but more so if teams were of a more technological-background). This suggests that when engineering RE and SA processes, a careful examination of the makeup of the team's non-technical properties should be done in order to determine which artefacts should be used during the process to maximize impact of intended effect.

3. Human factors such as mental capability, education, and others (e.g., professionalism, communication, leadership, etc.) significantly override the impact of technological usage in an RE and SA project.

A side result of S1 was that the individual mental capability of the architects was the prime factor for determining the final architecture quality (P3.2). Furthermore, other human-factors (such as professionalism, communication, etc.) impacted the "goodness" of RE and SA projects, but not necessarily to the same extent (a "moderate" co-relation with high-quality architectures – see Chapter 7 Section 3.3). However, these same non-technical factors had a high negative impact on the process itself – leading to extra time spent and rework, resulting in P3.1. In S5, participants also informally reported that a lack of coordination and communication mechanisms led to difficulties in enhancing and understanding the existing system (see Chapter 6 Section 3).

Despite employing varying available RE and SA technology, there was always a high variance in the productivity reported by various participants in the six studies, suggesting that the technology is not "streamlining" the process. In all academic studies (S1-S4, and S6), there was observed project variance that is invariably due to capability differences among project teams. The common technological support used by all teams in all studies did not decrease variance in teams' projects. These facts led to proposition P3.3.

Overall though, the contribution of mental capability and other non-technical factors to the results is not surprising. Numerous researchers have previously reported this in software development in general (Boehm, 1988; Curtis et al., 1988). These theoretical propositions are included in the theory for the sake of completeness and to provide a broader range of the lower-level theory.

3.2 Process factors theory

Level 2	**Level 1**	**Source**	**Key Terms**
P4. RE and SA artefacts and processes significantly vary when conducted in the presence or absence of an existing SA.	P4.1 Non-functional (NF) characteristics of a non-local sub-system significantly affect (enable or constrain) requirements for the local sub-system being worked on.	S4, S5	Existing SA effect on requirements products
	P4.2 Constrained requirements decisions have a (strong) negative impact on construction and testing.	S5	RE and SA interaction effects on software development processes
	P4.3 Constrained requirements decisions have a (moderate) negative impact on a multitude of project-specific system properties (such as performance, safety, reliability, etc.)	S5	RE and SA interaction effects on system

			quality
	P4.4 Older, "load-bearing" components of a system lead to more constrained effects on new requirements decisions than newer implemented components.	S5	RE and SA interaction effect factors
	P4.5 Significantly more *consequential* requirement decisions affect the SA than do core or emergent requirements decisions.	S5	RE and SA interaction effect factors
	P4.6 Requirements elicited from a RE process that involves analysis of a current architecture will be more technologically focused than a RE process that does not include such analysis.	S3	Existing SA impact on requireme nts product characteris tics
	P4.7 Requirements elicited from a RE process that does not analyze the current architecture will be more user-focused than a RE process that does not include such analysis.	S3	Existing SA impact on requireme nts product characteris tics
	P4.8 Requirements elicited when the current architecture is analyzed are considered more important for system success (in terms of providing essential value for system stakeholders) than without such analysis.	S3	Existing SA impact on requireme nts product characteris tics
	P4.9 Requirements elicited when the current architecture is analyzed are more architecturally relevant than requirements without such analysis.	S3	Existing SA impact on requireme nts product characteris tics
	P4.10 The degree of requirements characteristics will vary between projects, but the impact from presence/absence of SA will	S3	SA analysis impact

	be roughly the same.		factors
	P4.11 The existing architecture has a significant impact on new requirements decisions as a constraint or an enabler.	S4, S5	RE and SA interaction effect factors
	P4.12 Approximately 20-30% of requirements decisions are *constrained* by an existing SA.	S4, S5	RE and SA interaction effect factors
P5. The incorporation of RE and SA effect information into architecture impact analysis increases the effectiveness of RE and SA evolutionary processes.	P5.1 If the history of interaction effects between SA and RE is used effectively (in terms of cost-efficiency of the documentation, maintenance and retrieval of the critical RE and SA interaction information) then the time/effort spent performing evolutionary work in RE and SA will decrease.	S4, S5	Process support for RE and SA interaction effects
	P5.2 Retrieving pertinent lost RE and SA interaction information, from project sources such as people and existing project documents, for the purpose of making current RE and SA decisions increases RE and SA development time and cost.	S4, S5	Cost of lack of RE and SA interaction effect process support
P6. RE and SA processes that are augmented with sub-activities that enforce a tighter integration between critical RE and SA links will lead to an increase in the effectivenes	P6.1 Coordination between requirements analysts and architects during handover processes reduces number of problems during RE and SA activities.	S2, S5	RE and SA handover process support
	P6.2 Requirements-oriented problems encountered during architecting are predominantly limited to *quality satisfaction, quality drivers determination, modeling quality requirements, abstraction, and requirements understanding*.	S2	RE and SA process problems
	P6.3 If the requirements engineers and software architects together model quality requirements, then the number of requirements-oriented problems during the architecting process will decrease.	S2	RE and SA integration and process support
	P6.4 If adequate background information about	S2, S5	RE and

s and efficiency of these processes.	the requirements (such as, rationale, assumptions, priority, etc.) is given to, or shared with, the software architects then fewer requirements-oriented problems will be encountered by the architects.		SA handover process support
	P6.5 If the architects provide "live" feedback to the RE agents on potential system-wide constraints and enablers, then the amount of requirements-rework will be reduced.	S4, S5	RE and SA integration and process support

Table 8-3. Process factors theoretical propositions

4. RE and SA artefacts and processes significantly vary when conducted in the presence or absence of an existing SA.

The explicit focus of S3, S4 and S5 was investigating the issue of new requirements and requirements decision-making between two contexts: existing SA and no existing SA (or new systems development). Whereas S3 was strictly looking at differences in the resultant product (specifically the requirements), S4 and S5 examined requirements decisions made in-process and how they were affected by the existing SA. In S3, a set of propositions emerged directly based on the study's significant findings (P4.6 – P4.9).

P4.10 is not based on an explicit finding from S3, but arises based on the analysis of the generalizabiltiy of the results where project and domain factors will certainly influence requirements characteristics, but there is no reason why the specific characteristics that showed differences (such as *degree of technological focus, degree of user focus,* and *architectural relevance*) could not be consistent across projects. From this set of propositions from S3, the common theme is that there is a significant difference in a few of the resultant requirements products characteristics when elicited in the presence of an existing SA.

S4 and S5 differ from S3 in that they were investigating in-process requirements decisions in the presence of an existing SA. S4 was the initial study, and from its observations the set of propositions P4.1 - P4.2 were inferred. Based on these initial findings, S4 was replicated and extended in a larger-scale context (S5 – the RailCab project), where S5 probed deeper into the characteristics of the affected requirements decisions. It further led to new propositions (P4.3 – P4.6) that further demonstrate the impact of an existing SA on requirements decisions.

The quantitative results from S4 and S5 suggest that the RE and SA impact profiles will vary from project to project, and across domains. However, the same types of effects (e.g., constrained, enabled, none) do occur, and their impact on requirements decisions is significant, leading to proposition P4.11. There was a similarity in the frequency of the *constrained* effect type (23% vs. 30%) in both S4 and S5 which led to the derivation of proposition P12[48].

5. The incorporation of RE and SA effect information into architecture impact analysis increases the effectiveness of RE and SA evolutionary processes.

This proposition comes from the implications of the findings of S4 and S5, and not directly from the results. The essence of the proposition is that RE and SA processes should be augmented with process support (and any accompanying technology) for aiding architectural impact analysis in evolutionary processes (P5.1 and P5.2). These propositions have not been empirically observed or tested, but are more of a solution-oriented mechanism for dealing with specific phenomena that occurred in S3 and S4. Currently, in the research literature, RE and SA interaction effects are not explicitly incorporated in architecture impact analysis processes, and this emerging theory provides a framework for assessing the effectiveness of such information during RE and SA processes.

6. RE and SA processes that are augmented with sub-activities that enforce a tighter integration between critical RE and SA links will lead to an increase in the effectiveness and efficiency of these processes.

From the observations in S2, S4, and S5, problems were occurring during RE and SA that were as a result of a lack of process activities that enforced a tight integration on key links between RE and SA (P6.1 – P6.5). These problems were irrespective of the impact of human factors and so we conclude that these really are process specific

[48] Note that no other propositions in either Tables 2 and 3 have quantitative figures because they are not based on repeated studies. Thus, we opt for more conservative qualitative propositions which, if shown to converge with further studies (i.e., replications), can then be expressed quantitatively and the theory updated accordingly.

deficiencies; overcoming these deficiencies will maximize communication, coordination, and knowledge sharing on critical RE and SA issues that will in turn improve the effectiveness of an RE and SA process. The common theme among these propositions is that handover processes or RE and SA work focused on certain activities and artefact types need to be done in an integrated manner by the human agents conducting these processes. For other sub-activities and artefact types, the problems were not encountered. Knowing the minimal set of activities and artefact types that require "extra" care leads to more cost-efficient, yet still effective RE and SA processes.

3.3 Key Points

Overall, based on the key terms in Table 8-2 and Table 8-3, the proposed emergent theory states that:

> *The effectiveness of RE and SA processes is increased if technological support ensures:*
>
> *(1) tighter coupling between the artefacts and activities across RE and SA,*
>
> *(2) the project's development context (such and new development vs. enhancements, agile vs. traditional development models, centralized vs. distributed organization, etc.), and,*
>
> *(3) compatibility with the varying degrees of knowledge, skill-sets and personal motivation possessed.*

Here, the effectiveness of RE and SA processes is evidenced by such measures as higher quality of RE and SA products, lower development time, or reduced rework. Also, note that the theory statement is not arguing for always having a tighter process integration between RE and SA; as observed in S4 (see Chapter 4, Section 5.1) this is not always ideal. The theory is simply arguing that the employed technology has the capability for conducting both the RE and SA processes in an integrated manner wherever appropriate for a given project. This is orthogonal to the current design of industrial RE and SA tools where each tool focuses exclusively on only one of these processes (such as IBM's Doors for RE, and Rational's Software Architect for SA).

The theory statement is abstracted from the following higher-level propositions that are resultant from the key terms in Table 8-2 and Table 8-3:

- Different types of RE and SA knowledge, skill-sets and personal interests possessed by the human agents conducting RE and SA processes have a significant impact on RE and SA products and process quality, and also their characteristics.
- RE and SA processes should be augmented with sub-activities that enforce a tighter integration between critical RE and SA technical links in order to reduce the number of RE and SA problems encountered, and increase the coordination and knowledge sharing between RE and SA agents conducting these processes.
- RE and SA processes need process and technological support for capitalizing on the presence/absence of an existing SA; RE and SA product and process characteristics vary significantly between these two contexts.

We now present an initial evaluation of the emerging theory.

4 Evaluation of emerging theory

(Boehm and Jain, 2006) and (Sjoberg et al., 2008) list similar criteria for evaluating the "goodness" of theories, both lists of which were adapted for SE theory evaluation from other disciplines such as Business Management (Bacharach, 1989), Psychology (Haig, 2005), and Sociology (Cohen, 1989). The following criteria were amalgamated from the criteria in these two sources. Note that these criteria are all considered important and are thus their ordering does not indicate any priority.

1. Empirical support - The degree to which a theory is supported by empirical studies that confirm its validity.
2. Utility - The degree to which a theory supports the relevant areas of the software industry.

3. Generality - The breadth of the scope of a theory and the degree to which the theory is independent of specific settings.
4. Parsimony - The degree to which a theory is economically constructed with a minimum of concepts and propositions.
5. Testability - The degree to which a theory can be empirically refuted.
6. Explanatory power - The degree to which a theory accounts for and predicts all known observations within its scope, is simple in that it has few ad hoc assumptions, and relates to that which is already well understood.

For the rest of this section, the emerging theory will be evaluated with respect to each of the above criterion.

Empirical support:

Since this is an emerging theory, the number of empirical studies on which the theory is derived is low, and they are all from one research group creating a possible bias for the theory. Furthermore, each study investigated its own discrete topic; the propositions were not pre-determined to be directly related to each other (except for the ones involving S4 and S5) – other than them being loosely coupled in the context of RE and SA interactions.

A point in favour of the degree of Empirical support of the emerging theory is that, in fact, as shown in Table 8-2 and Table 8-3, there is a clear traceability from higher-level statements to lower-level statements, which are in turn based directly on empirical evidence from the studies. Except for P1.2 and P4.10, all statements are empirically grounded which is considered a stronger form of theory than more conjecture-based theory (Sjoberg et al., 2008).

Regardless, the empirical support is still considered low to moderate, which is expected in an emerging theory until more families of studies are conducted (Carver, 2010) in the domain of RE and SA interaction.

Utility:

The emerging theory can be directly used in the decision making in RE and SA projects with little adaptation, in particular for decisions concerning human and process factors. For example, concerning human factors, propositions P1.1 and P1.2 can be directly used in industrial hiring and training processes of relatively inexperienced requirements engineering or software architects, to ensure that the RE and SA staff have the appropriate types of knowledge for maximizing RE and SA performance. Likewise for process factors, proposition P4.4 can be used to inform new requirements decisions that the most problematic and costly decisions originate from the "load-bearing" subsystems, and thus extra attention should be given to these subsystems during impact analysis. Since all of the propositions have a direct practical impact (as in the above examples), we consider the utility of the theory to be high.

Generality:

The emerging theory's scope covers both human and process factors, and so is generalizable to these types of issues, irrespective of technology and setting. However, the theory does not extensively consider important RE and SA issues such as economic decisions, technological employment, organizational/project/team structure, and development lifecycle (e.g., agile vs. iterative) that would have an impact on the success of RE and SA projects. Furthermore, the empirical evidence on which the theory is derived is mostly from "lab" settings, which limits the generalizability of their findings, and thus any theory inferred from these studies. Therefore, the generality of the theory can be considered low to moderate.

Parsimony:

In the details of the studies (S1-S6), it is obvious that there is an expansive set of constructs of interest in the RE and SA interaction domain (e.g., specific RE and SA activities, products, types of human knowledge, etc.). In the emerging theory, this expansive set of constructs is reduced to a smaller, more manageable number that improve clarity and understandability of the theory, effectively leading to its easier application to practice. The Parsimony of the theory is thus considered high.

Testability:

Each proposition of the emerging theory is expressed in a way that is directly testable. Specific hypotheses can also be derived from the propositions; each proposition can essentially be tested in its own study with minimal dependencies on other propositions. Furthermore, the propositions are also testable through a variety of study designs and contexts; the most likely candidates for testing the propositions are industrial case studies and surveys, and "lab" controlled and case studies. The Testability of the theory is therefore considered high.

Explanatory power:

SE theories in general are considered to have difficulties with their explanatory power, due to the complexity of SE and the multitude of factors that can influence a SE project (e.g., human-based, technical, political, organizational, etc. [Boehm, 1988]). RE and SA processes are no different; they are complex processes that involve a wide range of factors (such as, heavy human involvement, conceptual artefacts, early part of a development thread, business and external issues, etc.) and so it is difficult to have a high degree of explanatory power for any particular theoretical concept. Since the theory presented in this chapter is emerging and is based mostly on observations from exploratory studies it provides first-step explanations for some of the aspects in RE and SA development (human and process related), but certainly further studies are required to expand and strengthen the explanatory power of the theory. The explanatory power is thus considered low.

5 Implications

There are numerous potential implications for the emerging theory for both practice and research which we list here.

- Practice:
  - Understanding the key activities in the RE and SA interaction - knowing these can aid in their smooth execution.
  - The impact of the existing SA on elicitation (e.g., enabled/constrained) and analysis (e.g., impact of non-functional attributes of the non-local subsystems on new requirements).

- Resource management (e.g., assignment of staff to projects and roles, training and hiring processes, etc.).
- RE and SA project planning.
- Predicting RE and SA project outcomes based on control/manipulation of key attributes to achieve desired business goals.

- Research:
  - Researchers can use preliminary theory for hypothesis forming and testing, which can then be fed back into the theory (i.e., transition from *descriptive* to *normative* theory [Sjoberg et al., 2008]).
  - Researchers can perform further grounded theory building on new research issues that were not explored in this dissertation (i.e., continue the preliminary inductive theory building established in this thesis).
  - Aids in assessing the maturity of the RE and SA interaction field (and its theory).

6 Conclusions

The RE and SA interaction discipline is abundant in underlying *construction* (or *engineering)* theory that is used to develop technology for aiding in the transitioning from RE to SA. However, "empirically grounded theory" on the interaction between RE and SA in terms of the *human* and *process* factors involved in RE and SA development is scattered and mostly anecdotal. We have thus presented an emerging theory that describes phenomena related to human and process factors in RE and SA development. The emerging theory is based on observations from the six studies reported in earlier chapters of this thesis.

The theory was constructed in a "bottom-up" manner and the chapter demonstrated clear traceability from the lower-level observations of the six studies reported in this thesis, to the higher-level propositions of the theory.

Furthermore, the emerging theory was evaluated based on "theory goodness" criteria from (Boehm and Jain, 2006) and (Sjoberg et al., 2008) and was found to satisfy the main criteria for a good theory (utility, generality, parsimony, testability, empirical support, and explanatory power) reasonably well, particularly considering the theory is based on a limited number of studies and is still at initial stages of development.

As with all theories, the emerging theory needs more empirical studies, either to test specific aspects of the theory, expand the breadth of propositions that are currently described, or to provide more detailed explanations as to why phenomena observed are occurring. This requires a concerted effort by the RE and SA community to conduct studies in various contexts (i.e., "lab" or industrial practice) and to feed the resultant findings back into the emerging theory.

References

Bacharach, S.B., 1989. Organizational theories: some criteria for evaluation. *Academy of Management Review*, 14(4): 496–515.

Boehm, B., 1988. A spiral model of software development and enhancement. IEEE Comp. Vol. 21, Iss. 5, pp. 61-72.

Boehm, B., Jain., A., 2006. An Initial Theory of Value-Based Software Engineering. In Value-Based Software Engineering 1st Edition; Eds: Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grünbacher G.; Springer, Germany; pp. 15-33.

Carver, J., 2010. Towards Reporting Guidelines for Experimental Replications: A Proposal. 1st International Workshop on Replication in Empirical Software Engineering Research (co-located with ICSE 2010), May 4, 2010, Cape Town, South Africa.

Cohen, B., 1989. *Developing Sociological Knowledge: Theory and Method*, 2nd edn, Belmont, CA, Wadsworth Publishing.

Curtis, B., Krasner, H., Iscoe, N., 1988. A Field Study of the Software Design Process for Large System. Comms of the ACM 31(11), 1268–1287.

Haig, B.D., 2005. An abductive theory of scientific method. *Psychological Methods*, 10(4): 371–388.

Jackson, M., 1994. The Role of Architecture in Requirements Engineering. Proceedings of the 1st International Conference on Requirements Engineering (RE '94'), pp. 241.

Shekaran, C., 1994. Panel Overview: The Role of Software Architecture in Requirements Engineering. Proceedings of 1st International Conference on Requirements Engineering, April 1994, pp. 239.

Sjøberg, D., Dybå, D., Anda, B. C., and Hannay. J., 2008. Building theories in software engineering. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, Guide to Advanced Empirical Software Engineering, pages 312–336, Springer.

Software Requirements to Architectures Workshop (STRAW), 2001. International Conference on Software Engineering (ICSE) workshop, June 2001, Toronto, Canada.

Software Requirements to Architectures Workshop (STRAW), 2003. International Conference on Software Engineering (ICSE) workshop, May 2003, Portland, USA.

Chapter 9

Conclusions and Future Work

We state the conclusions of the thesis before discussing future work.

9.1 Conclusions

While the interaction between RE and SA has been recognized and researched in terms of new technology (methods [Bass et al., 2003][Wang et al., 2005], development methodologies [Castro et al., 2002], tools [Bachmann et al., 2003a], processes [Schwanke, 2005][Brandozzi and Perry, 2003], etc.), there is a distinct lack of empirical understanding regarding the scientific properties of this interaction (such as: the impact of an existing SA on newly elicited requirements, how non-technical factors affect RE and SA processes, and the specific architecting process areas that architects without RE knowledge encounter the most difficulties). Furthermore, in SE, both technical and human aspects are considered critical for the success of software development (John et al., 2005; Bass and Berenbach, 2008; Clements et al., 2007). In particular, for RE and SA, human factors are even more important due to these processes lying at the front-end of the development cycle and therefore being more aligned with real-world issues. Thus, the scientific properties of the RE and SA interaction can be broken down into two key areas: human and technical based.

To address this lack of scientific understanding, we conducted a suite of six empirical studies over the past five years, each addressing a distinct topic under the general area of human and technical factors in the interaction between RE and SA. Examples of such issues addressed are: what are the critical types of RE and SA knowledge required to produce high quality RE and SA products? How does requirements decision-making change when done in the presence of an existing SA? What types of requirements-oriented problems occur while architecting?

These six studies are documented, each in its own specific chapter, and were conducted in a variety of contexts, such as academic "lab" experiments and studies, and case study on a real large-scale project. The studies also employed a mix of empirical methodological designs, from quantitative based data collection and analysis, to more Social Sciences oriented qualitative techniques (Creswell, 2003).

Based on the observations from these six studies, an emerging grounded theory is proposed that describes the impact of human and technical factors in the interaction between RE and SA. In short, the emergent theory states that:

The effectiveness of RE and SA processes is increased if technological support ensures:

(1) tighter coupling between the artefacts and activities across RE and SA,

(2) the project's development context (such and new development vs. enhancements, agile vs. traditional development models, centralized vs. distributed organization, etc.), and

(3) compatibility with the varying degrees of knowledge, skill-sets and personal motivation possessed.

Further to this theory, specific findings from the various studies were discovered that add to the body of knowledge in RE and SA. From this, we conclude that:

1) software architects with RE knowledge performed approximately 16% better than architects without such knowledge (see Chapter 3);

2) a third of all problems encountered during architecting are requirements-oriented. The highest severity requirements-oriented areas are: quality satisfaction, quality drivers determination, modelling quality requirements (scenarios), abstraction, and requirements understanding (see Chapter 4).

3) that requirements elicited in the presence of an existing SA are more technologically-focused, architecturally relevant, and important. Conversely requirements elicited without an existing SA are more user-focused (see Chapter 5);

4) that there are three ways in which an existing architecture affects new requirements decisions: *constrained*, *enabled*, and *none* (see Chapters 6 and 7);

5) that approximately 40-50% of new requirements decisions are affected by the existing SA (see Chapters 6 and 7);

6) that approximately 20-30% of new requirements decisions are *constrained* by the existing architecture (see Chapters 6 and 7);

7) that most of the *constrained* requirements decisions originate from implementation activity feedback (see Chapter 6);

8) that early architecting decisions in "load-bearing" subsystems are the source of the most costly and problematic new requirements decisions (see Chapter 6);

9) that non-technical factors have a moderate impact on RE and SA products, and a high impact on RE and SA processes (see Chapter 7);

10) and that the representation of non-technical skills in the current ACM SE and CS curriculums are under-represented for effectively performing RE and SA processes (see Chapter 7).

While our exploratory findings and theory are promising (see their implications in Chapters 2-8), they are but the start for developing a mature scientific discipline (Sjoberg et al., 2008). The results are thus limited to the contexts of the study and their use in out-of-context areas is cautioned. Further, only through the execution of families of studies will the body of knowledge solidify (Carver, 2010), leading to improved scientific understanding that can reliably be used in wide practice (Kitchenham et al., 2004). We encourage others in the research community and industry to join in this quest.

9.2 Future Work

The opportunities for further work centre on two orthogonal dimensions: technology development and further RE and SA interaction empirical work, which we now discuss.

9.2.1 RE and SA technology development

The suite of exploratory empirical studies presented in this thesis focused on "new knowledge" or "problem exploration" and were not directly solution-oriented. However, with the various issues observed in the six studies, we believe new avenues of technological research can be researched to aid RE and SA practitioners.

For example, in the more human factors based studies (Chapters 2-3 and Chapter 7) we observed that a lack of specific types of knowledge during RE and SA processes (such as RE or SA knowledge) had a significant detrimental impact on RE and SA quality (product and process). RE and SA decision support tools support could then possibly be developed that encode key RE and SA knowledge, and using this knowledge, aim to guide RE and SA practitioners through the more difficult process activities when making RE and SA decisions. Essentially, these tools could help to bring the RE and SA practitioners to a minimum baseline instead of relying entirely on the competence and knowledge-level of the RE and SA practitioners. The ArchE tool by (Bachmann et al., 2003b) is an example of a first step in this direction, as they guide practitioners through determining and analysing a few architectural tactics (*modifiability* and *performance*), but more work is needed to encompass the wider range of RE and SA activities and artefacts.

Another aspect that can possibly be incorporated into an RE and SA decision support tool is the integration of RE and SA interaction effects as observed in Chapters 4 and 5. Currently, there are few tools that allow for concurrent RE and SA design, and these tools do not focus on issues such as impact analysis during RE. For example, while the RE and SA practitioners are documenting their requirements, tool support that showed which parts of the architecture historically affected these requirements and what implementation implications there were would be invaluable for making cost-effective requirements decisions. However, the real research challenge here would be in capturing and maintaining this key RE and SA interaction information in a cost-efficient manner, either through automatic means or guiding the users towards eliciting a small subset of what they deem is the critical information, and suitably enforcing that this information is maintained on an ongoing basis. The participants in the RailCab study (described in Chapter 6) regularly mentioned that having such information would be useful. In particular, when staff turnaround occurred, they often neglected periphery information capture because this incurred a short-term time penalty that posed a risk to the completion of impending deadlines.

Beyond tool support, current RE pedagogical textbooks do not describe how to do RE in the presence of an existing SA; impact analysis techniques are virtually non-existent yet most industrial projects are of a more maintenance nature (*i.e., brownfield development)* and are not *greenfield* as implicitly assumed in current RE literature. Future technological research could then focus on expanding current methods and techniques to add support in how to more precisely determine cost and feasibility of new requirements due to technical constraints or enablers from the existing architecture (see Chapters 5 and 6 for more details on these factors). Conversely, architecting methods and techniques could be updated or redesigned to add support for architecting during maintenance phases of the software lifecycle.

9.2.2 Further RE and SA interaction empirical work

The empirical suite of studies presented in this thesis, along with the emerging theory, provides an initial body of knowledge on the scientific properties of the interaction between RE and SA processes. Despite the promise of this new knowledge and theory, the empirical studies conducted were all exploratory, and are thus just the beginning for establishing a solid grounded body of knowledge. What is needed then are further replications and extensions to these studies, as well as testing of the studies emergent hypotheses. Specifically, these studies need to be conducted in a variety of contexts and settings, on projects with different degrees of complexity, and in different domains to assess which results are common among the studies, and which results do not hold. These findings would then be fed back into the theory to add, modify or delete current theoretical propositions that have been proposed.

Further to this general replication, the studies, when replicated, should be extended with new or expanded research questions (where possible) that probe deeper into explaining the phenomena observed in this thesis. As discussed in Chapter 8: Emerging theory, due to the emerging nature of the theory, the theory's explanatory power is low and requires further studies that go beyond the *exploratory* nature of these studies (which focus more on breadth of observations [Mason, 1996]) to more *explanatory* based studies (which focus on depth of observations [Runeson and Host, 2009]).

References

Bachmann, F., Bass, L., Klein, M., 2003a. Moving from quality attribute requirements to architectural decisions. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 122-129.

Bachmann, F., Bass, L., Klein, M., 2003b. Preliminary Design of ArchE: A Software Architecture Design Assistant. Technical Report, Software Engineering Institute, Carnegie Melon University, CMU/SEI-2003-TR-021 ESC-TR-2003-021.

Bass, L., Berenbach, B., 2008. Leadership and management in software architecture (LMSA'08). a report on an ICSE workshop. ACM SIGSOFT Software Engineering Notes 33(4): 27-29.

Bass, L., Clements, P., Kazman, R., 2003. *Software Architecture in Practice, 2nd edition*, Addison-Wesley.

Brandozzi, M., Perry, D. E., 2003. From Goal-Oriented Requirements to Architectural Prescriptions: The Preskriptor Process. Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 107-113.

Carver, J., 2010. Towards Reporting Guidelines for Experimental Replications: A Proposal. 1st International Workshop on Replication in Empirical Software Engineering Research (co-located with ICSE 2010), May 4, 2010, Cape Town, South Africa.

Castro, J., Kolp, M., Mylopoulos, J., 2002. Towards Requirements-Driven Software Development Methodology: The Tropos Project. Information Systems, June 2002.

Clements, P. C., Rick Kazman, Mark Klein, 2007. Working Session: Software Architecture Competence. WICSA 2007.

Creswell, J. W., 2003. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* Thousand Oaks, CA: Sage Publications.

John, M., Maurer, F., Tessem, B., 2005. Human and social factors of software engineering: workshop summary. ACM SIGSOFT Software Engineering Notes Volume 30, Issue 4 (July 2005), pp. 1 – 6.

Kitchenham, B., A., Dyb □a, T., Jøorgensen, M., 2004. Evidence-based software engineering. In ICSE '04: Proceedings of the 26th International Conference on Software Engineering., pages 273–281, Washington, USA.

Mason, J., 1996. Qualitative Researching. SAGE Publishing Ltd., London, England.

Runeson, P., and Host, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Journal of Empirical Software Engineering, Vol. 14, #2, pp. 131-164.

Schwanke, R., 2005. GEAR: A Good Enough Architectural Requirements Process. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, USA, pp.57-66.

Sjøberg, D., Dybå, D., Anda, B. C., and Hannay. J., 2008. Building theories in software engineering. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, Guide to Advanced Empirical Software Engineering, pages 312–336, Springer.

Wang, Z, Sherdil, K., Madhavji, N., 2005. ACCA: An Architecture-centric Concern Analysis Method. IEEE Working International Conference on Software Architecture (WICSA), Pittsburgh, USA, November 2005, pp. 99-108.

Appendix A

Software Architecture Assessment Instrument[49]

Assessor Name(s): ______________________________
Project Team #____

—

Purpose: The purpose of this instrument is to help assess the Software Architecture projects. The assessments will be used in an empirical study on software architectures.

Background: Bredeyemer consulting[50] states that a Software Architecture should be:

- *Good* — i.e., *it is technically sound and clearly represented.*
- *Right* — i.e., *it meets the needs and objectives of key stakeholders.*
- *Successful — i.e., it is actually used in developing systems that deliver strategic advantage.*

This assessment instrument is meant to assess the Architecture with respect to only the first two attributes listed since the architectures in question will not be implemented, or judged, in any real-world setting. The *minimum* amount of work that the instrument is based on is taken from the document "Minimum Project Documentation". This document was given to the students to state the expected type and quantity of the various architecting artifacts that should be in the final product. Also, the students used documentation templates from the course textbook (Bass et al., 2003) to complete their project so these templates are also a source for this instrument.

Instructions:

Use the accompanying Microsoft Excel template to rate each statement according to the following scale except where otherwise instructed. Each statement refers to the 'level of agreement' of the statement. The scale is: 6 – very strongly agree; 5 – strongly agree; 4 – mostly agree; 3 – neither agree nor disagree; 2 – mostly disagree; 1 – strongly disagree; 0 – very strongly disagree. The template also has three columns for "Evaluator Confidence", "Rationale", and "Suggestions". In the

[49] The format and style of this instrument is based on an assessment manual from the field of Psychology. J. Fortin and C. Cuerrier: *Evaluating a Mentoring Program,* Canadian Cataloguing in Publication Data, 2003.
[50] Ruth Malan and Dana Bredemeyer: "The Visual Architecting Process™", Architecture Resources for Enterprise Advantage, http://www.bredemeyer.com, 2003.

confidence column, mark your confidence level from 1-10 (where 1 is very little confidence and 10 is extremely confident) for each of your responses. In the rationale box, provide where necessary a justification to your rating. The suggestions column is where you can input feedback about the instrument for any given statement.

1. Domain Work

The domain work includes tasks that are based on understanding the requirements and the domain of the system. They deal with issues that are not part of the design of the system, but more with modelling and understanding the *problem definition* of the system to be built.

1.1 The context diagram(s)

1. *Minimum one context diagram showing overall system within its environment (2 – Has more than one context diagram for different sections, 1 – meets this requirement, 0 – no context model).*
2. *Models show links to all external institutions and entities in the problem domain (Completeness) (should be 4 or 5 links in the ideal solution).*
    - *The Banking system.*
    - *ATM, internet phone banking, direct staff access, and automated phone banking.*
    - *Other financial institutions such as: other banks, stock exchange, government institutions.*
    - *Other users such as managers and maintainers.*
3. *Model(s) (possibly explained by supporting description) are easy to read and understand.*
4. *The context model(s) are too complex and detailed for the level of abstraction they are representing.*

1.2 Use Cases (Enter N/A if there is no work on the use cases)

1. *Existence of use cases for key functionality such as withdraw and deposit funds, transfer funds, check account balance, and edit personal information.*
2. *Clear and logical models.*
3. *Use cases are rooted in the requirements.*
4. *The use case models, components, or links are superfluous.*
5. *Appropriate labelling of links between elements in the models.*

2. Requirements-Architecture Work

Bass, Clements and identify the quality scenario work and tactics determination to be the tasks that lie in the link between Requirements and Architecture. Other researchers have proposed other RE-SA methods that would involve a different set of RE-SA tasks, but since the subjects of our study used Bass et al.'s ADD process, we are using their definition of RE-SA tasks.

2.1 Quality work (attributes and scenarios) (Enter N/A if there are no quality scenarios).

1. *Explicit quality scenarios that are representative of the problem domain.*
2. *For each scenario, six elements of quality scenarios are required: (for each of the elements of a scenario, give a '1' if it exists and is reasonable, and a '0' if it is not. Total will be given out of six)*
    - *Source*
    - *Stimulus*
    - *Environment*
    - *Artefact*
    - *Response*
    - *Response measure*

3. Architecting

Architecting forms the bulk of the tasks the subjects had to perform. It involves high-level design focused on creating the structure of the system, and the relationships within this structure. Everything from the conceptual models to the corresponding documentation is included in Architecting tasks.

3.1 Architectural Structure

3.1.1 Module-level view

"In the module view, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. There is less emphasis on how the resulting software manifests itself at runtime. The module structure allows us to answer questions such as, "What is the primary functional responsibility assigned to each module? What other software elements is this module allowed to use? What other software does it actually use?"

1. *Appropriate use of architectural patterns (either selected or created) to satisfy quality attributes and tactics. Please refer to BCK (chapters 3-7) textbook for discussion on patterns and how they relate to quality attributes and tactics*
2. *<u>Three</u> levels of decomposition for the main functions: (3 – three levels, 2 – two levels, 1 – 1 level, 0 – no levels)*
    - *deposit (cheque or cash)*
    - *withdraw*
    - *check balance*
    - *transfer funds between accounts*
    - *view/print recent transactions*
    - *pay bills*
    - *edit personal information*

3. *At least two levels of module decomposition for "other" features such as:*
    - *3.1 user interface functionality: (2 – two levels, 1 – one level, 0 – no level)*
        - *Representation of differences between ATM, Internet Banking, Staff Access, and Telephone Banking.*
    - *3.2 security features: (2 – two levels, 1 – one level, 0 – no level)*
        - *Encryption/decryption, authentication, and audit trail, among others.*
4. *At least one level for database functionality (1 – yes, 0 – no).*
5. *The different levels of decomposition are consistent with one another:*
    - *5.1 Features well-defined modules whose functional responsibilities are allocated on the principles of information hiding and separation of concerns.*
    - *5.2 Sub-modules coverage of the parent module functionality.*
    - *5.3 Sub-module links show the data flowing up/down the links (see Sommerville's SEng book* (Sommerville, 2006) *on module decomposition).*
    - *5.4 The sub-module dataflow (in/out) are consistent with the in/out dataflows of the parent (Sommerville, 2006).*
6. *Diagrams are readable (not "messy").*
7. *Elements in a model are given appropriate names.*

3.1.2. Deployment view

"This view is meant to show the relationship between the software elements and the elements in one or more external environments in which the software is created and executed. They answer questions such as:

- What processor does each software element execute on?
- In what files is each element stored during development, testing, and system building?
- What is the assignment of software elements to development teams?"

1. *Minimum one deployment structure for a particular level of decomposition (2 – goes beyond the required one section, 1 – one section, 0 – none).*
2. *Appropriateness of the patterns selected/created with respect to the quality attributes and tactics.*
3. *Deployment view is centred on appropriate issues (such as, network topology, assignment of software units to processors, middleware, etc.) based on fulfillment of quality attributes (1 – yes, 0 – no).*
4. *Understandibility – the model is conceptually clear.*
5. *Readability – the model is well-labelled and clear, use of key for notation.*
6. *Logical displacement and labelling of links (relations) between the elements.*

3.1.3 Component and Connector (C&C) view

"In this view, the elements are runtime components (which are the principal units of computation) and connectors, which are the interactions among the components. Component-and-connector structures help answer questions such as:

- What are the major executing components and how do they interact?
- What are the major shared data stores?
- Which parts of the system are replicated?
- How does data progress through the system?
- What parts of the system can run in parallel?"

1. *Minimum one C&C structure section for a particular level of decomposition. (2 – goes beyond the required one section, 1 – one section, 0 – none)*
2. *Appropriateness of the patterns selected/created with respect to the quality attributes and tactics.*
3. *C&C view is centred on an appropriate issue based on fulfillment of quality attributes (e.g. showing concurrency for performance, timing properties, data-flow, etc.). (1 – yes, 0 – no)*
4. *Understandibility – the model is conceptually clear.*
5. *Readability – the model is well-labelled and clear.*
6. *Logical displacement and labelling of links (relations) between the elements.*

3.2 Overall Architecture

1. *Buildability:*
    a. *Architecture amenable to be assigned to separate development groups for implementation and, subsequently, amenable to incremental integration and incremental testing.*
2. *The architecture depends on a specific version of a commercial product (1: yes, 0: no)*
    a. *If yes, architecture is structured so that changing to a different product is straightforward and inexpensive.*
3. *The various views (module, C&C, and deployment) all map to each other in a seamless, non-conflicting way. They depict different aspects of the system.*

3.3 Documenting an Architecture

3.3.1 Interfaces

1. *Completeness – has interface description for the lowest levels of decomposition of the main features listed below (put a 1 for exist, 0 for not described).*
    a. *Print Reports for bank manager*
    b. *withdraw money*
    c. *deposit money*
    d. *transfer funds*
    e. *check balance*
    f. *cancel card*

   g. *postdate transactions*
   h. *pay bills*
   i. *order cheques and bonds*
   j. *edit personal information*
   k. *request stop payments*
2. *Interface description includes public services of a module.*
3. *Interface description includes information an element needs in order to perform a function.*
4. *Interfaces are accurate and correct to the extent they can be at this level of abstraction.*
5. *Interfaces are written in a format (can be any) that is readable and understandable.*

3.3.2 Behaviour

1. *All the* critical *functionality is represented using sequence, state, and/or activity diagrams (completeness).*
2. *Behaviour diagrams maintain consistency with the rest of the models representing the" architecture?*
3. *Redundancy (functions that are similar in behaviour, such as withdraw, deposit and check balance for each of the different types of access (ATM, internet banking, etc.), are not represented multiple times).*
4. *Models are technically correct.*
5. *<u>Architectural</u> behaviour is depicted (meaning the behaviour across elements, <u>not</u> within a given element) (3 – all the time, 2 – most of the time, 1 – very little, 0 – no architectural behaviour).*
6. *Diagrams are clear and easy to read.*

3.3.3 Descriptions

This section is for the textual descriptions of the views. The elements and their relations are described to complement the graphical models that are given.

1. *Sound grammar and spelling.*
2. *Describes enough information to understand the system at this level of abstraction.*
3. *Completeness – existence of descriptions of all elements and their relations.*

3.3.4 Architecture Background (rationale, assumptions, analysis of results, and design alternatives)

This section contains all the reasoning description about the corresponding sections of the architecture. Items such as rationale, assumption, analysis of results, and design alternatives should all be detailed in this section.

1. *Rationale is based on quality attributes trying to achieve and the means for achieving them.*

2. *Quality attribute tradeoffs and sensitivity points are made explicit.*
3. *Discusses possible design alternatives and why they were dismissed.*
4. *Sound grammar and spelling.*
5. *Appropriate explicit assumptions should be documented for each view.*

3.3.5 Overall Documentation

1. *Existence of: (enter 1 for yes, 0 for no for each of the following documentation elements)*
   a. *Page numbering*
   b. *Table of contents*
   c. *Section headers*
   d. *Glossary of terms*
   e. *References are used when necessary*
2. *Documentation across views section (see pages 215-218 in the course textbook (Bass et al., 2001).*
3. *Models have key to describe the notation used for modelling.*
4. *Consistency of documentation across all sections. Different individuals might be responsible for different sections of the documentation, so is there differences in the format, structure, or writing style of the various sections?*
5. *The documentation is well structured, organized and clear.*

Appendix B

Requirements Ratings Data Collection Instrument

The requirements ratings data collection instrument was administered to each requirements rater to collect the requirements rating data (see Section 3.6.1). The instrument was operationalized through an MS Excel Spreadsheet file, and the organization of the spreadsheet is organized from the structure of Table 1 below.

Essentially, each requirement takes up two rows of this table. The first row is where the information pertaining to the requirement is given to the raters, and where they enter the ratings for the different requirements characteristics. Specifically, for each requirement, there are four pieces of information given to the reviewer: requirements ID, a title, a description and a rationale. The requirement ID is a numerical value that uniquely identifies the requirement. The title explicitly indicates what part of the system the requirement is referring to (Tele-Banking, Wireless Banking, Web Banking or Interac.) The description is the requirement itself, and the rationale provides additional reasoning as to why the requirement is necessary. These four pieces of information are given in the first four columns of the table. The next twelve columns[51] are where the rater enters their rating for the particular requirements characteristic given in the column header. The raters filled out this part of the instrument with reference to the list of requirements characteristics, their definitions and the scales to use for each characteristic (see Table 1). In the second row for a given requirement, the rater can optionally leave any comments regarding their specific ratings for a particular requirement characteristic entry.

Note that in order to remove possible *researcher bias* during the ratings process, the table does not contain any information that can associate given requirements with specific teams that elicited the requirements, and whether they had access to the existing SA during their RE project.

The results of each individual rater's assessment are merged into another MS Excel sheet which is organized based on the structure from Table 4-2 in Chapter 4: Section 3.6.1, where the inter-rater agreement procedure from Section 3.6.1 can be conducted.

[51] Due to readability of the template, not all columns with characteristics are shown. See Table 4-1 for full list.

Table 1. Requirements Rating Data Entry Template

Requirements Information				Requirements Characteristics					
Reqt. ID	Title	Description	Rationale	Cost	Time	Quality	User Needs	Tech. Needs	... (cont'd.)
R1									
Rater comments									
R2									
Rater comments									
R3									
Rater comments									