Available online at www.sciencedirect.com

ScienceDirect

The Journal of Systems and Software 81 (2008) 1470–1490

The Journal of Systems and Software

www.elsevier.com/locate/jss

Software architecting without requirements knowledge and experience: What are the repercussions? ☆

Remo Ferrari, Nazim H. Madhavji *

Department of Computer Science, University of Western Ontario, Middlesex College, London, ON, Canada N6A 5B7

Received 7 September 2007; accepted 15 December 2007
Available online 17 March 2008

Abstract

Whereas the relationship between Requirements Engineering and Software Architecture (SA) has been studied increasingly in recent years in terms of methods, notations, representations, tools, development paradigms and project experiences, that in terms of the human agents conducting these processes has not been explored scientifically. This paper describes the impact of requirements knowledge and experience (RKE) on software architecting tasks. Specifically, it describes an exploratory, empirical 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 software 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. This paper identifies specific areas of both architecture design as well as the architecture-development process where the differences manifest between the RKE and non-RKE architects. The paper also describes the possible implications of the findings 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 the Appendix) for quantitative analysis of the quality of a software architecture. This paper also describes some new threads for future work.

Keywords: Software Architecture; Requirements knowledge and experience; Software quality; Architectural assessment instrument; Attribute Driven Design (ADD) method; Hiring and training; Software engineering curriculum; Architecture and requirements technology; Quantitative and qualitative research; Empirical study

1. Introduction

The relationship between Requirements Engineering (RE) and Software Architecture (SA)[1] 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–SA transitioning (STRAW 01, STRAW 03) and some individual research works, such as (Damian and Chisan, 2006; Wang et al., 2005; Schwanke, 2005; Poort and De With, 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–SA relationship in this manner because, not least

☆ A preliminary version of this paper was published in (WICSA, 2007) (Ferrari and Madhavji, 2007).

* Corresponding author. Tel.: +1 519 661 3561; fax: +1 519 661 3515.
E-mail addresses: rnferrar@csd.uwo.ca (R. Ferrari), madhavji@csd.uwo.ca (N.H. Madhavji).

[1] For the rest of the paper, the acronym SA will refer to: Software Architecture as a discipline, a software artefact, or the architecting process. The context in which the acronym appears dictates its meaning.

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–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 exist to our knowledge that critically examine the RE–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., 2003).

Some critical questions in the RE–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 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–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 10 respected institutions that offered SA, only one SA course had a dedicated RE course as a pre-requisite; three had only a general software engineering (SE) course as a pre-requisite; and the rest had no (SE or RE) pre-requisites. Also, the recent IEEE/ACM curriculum for SE (Software Engineering, 2004) recommends general SE or software construction as pre-requisites for SA courses, depending on the core package selected. Thus, our findings could be a trigger for possible streamlining of the pre-requisites 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 15 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[2] of RE and SA suggest that only 15% were in the above mentioned categories combined. This status of research suggests that studies such as 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 categorised based on various properties such as whether they should be implemented as components, bus, system properties, and so on. Liu et al. (2003) extend this method by introducing a rule-based framework that allows for requirements-architecture mappings to be automated where possible. Liu 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. (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 (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 organisation, 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

[2] We examined 552 papers from the Requirements Engineering Journal (years 2000–2007), IEEE International Requirements Engineering Conference (RE) (years 2000–2006), IEEE/IFIP Working International Conference on Software Architecture (WICSA) (years 2000–2007), 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).

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 organising 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 and De With (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. (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) investigate the interaction between requirements and software architectures. Specifically, they explore the effect of software architecture documentation 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 requirement 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 software 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 analyzed 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 in Section 4.1) 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 centred 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 sub-section, the variables of interest are introduced and discussed, along with their associated metrics (see Table 1).

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*[3] is used when the main property of interest (in our case, RKE vs. non-RKE) is *innate* to the participants *before* the study is conducted.[4]

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

[4] 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.

Table 1
Summary of study variables

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

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 15 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.

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 85 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 15 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 timelogs.

Table 2
Data collection templates

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

The qualitative data was dominated by emails as well as feedback sessions that constituted approximately 50 h 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 for more details).

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*[5] 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[6] 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.

[5] *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.

[6] QSR NUD ∗ IST 4.0. QSR International Pty Ltd., 1999. Available at http://www.qsrinternational.com.

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.*
- "*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, 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, 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 et al., 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 software 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, 2003; 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. Another ecological validity threat is that the project focused on a single, banking, domain and so we should not assume that the results would be generalizable to all application domains. Still, to contain this threat, we intentionally used a domain that appears to be no less complex than other domains. Yet another threat is due to the architecting and requirements processes involved in the study. We note that the concepts and elements in these processes that were tracked for impact analysis are germane to those typically found in interative processes used in the industry. For example, we avoided the analysis of particular types of encoded models (such as UML diagrams). Thus, in general this threat is contained.

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 analyzed 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.

3.9. 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 software architectures than do architects without RKE.*

Before conducting a statistical test, we set the alpha level (or level of *significance)* to be .05.[7]

This hypothesis was analyzed and tested using analysis of covariance (ANCOVA[8]) 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*[9] *of feedback* (see Section 3.7.2), 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 3 presents the *means* (*as well as other descriptive statistics*) of the architecture quality of the two groups (RKE and non-RKE).

In Table 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 $p = 0.076$ 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 effort. Without such further analysis, a statistically significant result cannot be determined.

Looking at the other variables of interest, in their respective academic backgrounds there was a 1-point difference between them (72.9 for non-RKE, 73.9 for RKE), 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

[7] In some exploratory studies, the confidence interval set can be lower, as low as 0.1 or 0.2, (WebStat, 2000). However, we felt that by setting the alpha level to be .05, we reduce the risk of having a Type 1 error (false positive).

[8] 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.

[9] 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.

Table 3
Descriptive statistics for study variables

	Descriptive statistics	Architecture quality	Academic background	Feedback	Effort
Non-RKE	Mean	62.4	72.9	65.4	139.0
	Standard deviation	16.5	4.0	44.0	34.5
RKE	Mean	78.5	73.9	51.3	139.8
	Standard deviation	15.7	4.4	30.1	41.0
Total	Mean	69.9	73.4	58.8	139.4
	Standard deviation	17.6	4.1	37.5	36.3

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 on the architectural quality variable as the dependent variable (see Table 1), the results of which are shown in Table 4.

In Table 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–see Table 3), especially considering the academic background of both types of groups was approximately equal (See Table 3 means for Academic Background). 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 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.*[10] It is not that the Feedback 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, the 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.[11] 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 are

Table 4
ANCOVA table for architecture quality

Source of variation	DF	Sum of squares	Mean square	*F*	Sig.
Academic background	1	1848.238	1848.238	15.431	0.003
Feedback	1	118.667	118.667	0.991	0.343
Effort	1	35.095	35.095	0.293	0.600
RKE	1	713.902	713.902	5.96	0.035
Error	10	1197.727	119.773		
Total	15	77590.815			

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

[11] 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.

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 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.

The three bar charts (Figs. 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.

Table 5
Average feedback per team

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

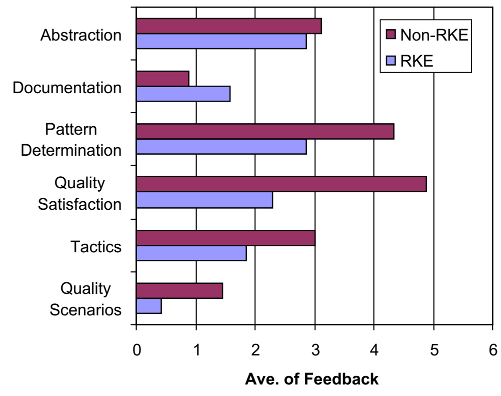

Fig. 1. Difference of "Hints".

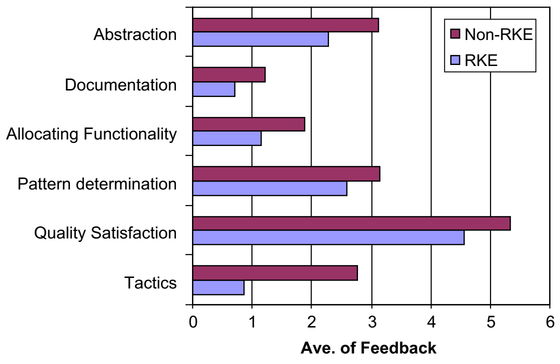

Fig. 2. Difference of "Explanations".

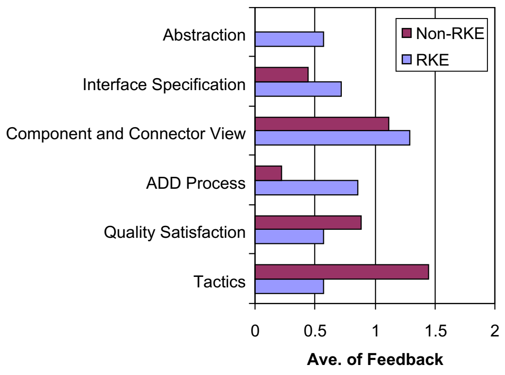

Fig. 3. Difference of "Give Aways".

4.2.1.1. Hints.

Fig. 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. This finding gives substance to Bachmann et al.'s (2003a) intuition 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.

Fig. 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. 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:

RE (**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 Section 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. Fig. 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.[12]

Table 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 ($p = 0.035$, see Section 4.1 for more detail). We now analyse below the results from the detailed assessment of each topic.[13]

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 *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 Figs. 1–3) 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 (understandability, 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.

[12] 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.

[13] 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.

Table 6
Architecture quality comparison

Architecture topics	RKE average (out of 100)	Non-RKE average (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

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 Figs. 1–3), 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.*

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 pre-requisite 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 scores on *architectural descriptions* are slightly higher for the RKE group than for the non-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 sub-sections 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, Ferrari and Madhavji (2006) 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 Section 1, 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 pre-requisites 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 pre-requisite (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., 2001), 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–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:

- In Section 4.1, we stated the following hypothesis:

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

  This hypothesis provides a solid basis to conduct a replicated study 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 generalization 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 Software 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 01 and STRAW 03). 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% (see in Table 3, RKE: 78.4% and non-RKE: 62.4%). 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 RE 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 software 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, software 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 software 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 based on the emergent hypothesis stated in Section 4.1.; examining how requirements that evolve during architecting affect the RE–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 generalize 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.

Acknowledgement

This work was, in part, supported by Natural Science and Engineering Research Council (NSERC) of Canada.

Appendix. Software Architecture Assessment Instrument[14]

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[15] 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 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

[14] 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.

[15] Ruth Malan and Dana Bredemeyer: "The Visual Architecting Process™", Architecture Resources for Enterprise Advantage, http://www.bredemeyer.com, 2003.

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. *Threelevels 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 onelevel 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. *Understandability – 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. *Understandability – 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. *post-date 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. *Architectural behaviour is depicted (meaning the behaviour across elements, notwithin 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., 2003*).*
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.*

References

Bachmann, F., Bass, L., Klein, M., 2003a. Moving from quality attribute requirements to architectural decisions. In: 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, second edition. Addison-Wesley.

Berander, P., 2004. Using students as subjects in requirements prioritization. In: 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. Pearson Allen & Bacon, Boston.

Boehm, B., 2002. Get ready for agile methods, with care. Computer 35 (1), 64–69.

Brandozzi, M., Perry, D.E., 2003. From goal-oriented requirements to architectural prescriptions: the preskriptor process. In: 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. Architecture Resources for Enterprise Advantage. Bredemeyer Consulting.

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

Carmines, E.G., Zeller, R.A., 1991. Reliability and Validity Assessment. Sage Publications, Newbury Park.

Carver, J., Jaccheri, L., Morasca, S., 2003. Issues in using students in empirical studies in software engineering education. In: 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. In: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA '07), Mumbai, India.

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

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.

De Vaus, D.A., 2002. Analyzing Social Science Data. SAGE Publishing Ltd, London.

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), 433–453.

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. In: 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. In: First International Workshop from Software Requirements to Architectures (STRAW '01), Toronto, Canada.

Ferrari, R., Madhavji, N.H., 2006. Requirements-oriented problems while architecting: an empirical study. In: 12th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06), Luxembourg, pp. 81–96.

Ferrari, R., Madhavji, N.H., 2007. Impact of Requirements Knowledge and Experience on Software Architecting: An Empirical Study. In: 6th Working IEEE/IFIP Conference on Software Architecture (WICSA '07), Mumbai, India.

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, 201–214.

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

In, H., Kazman, R., Olson, D., 2001. From requirements negotiation to software architectural decisions. In: 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. In: 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. In: Second International Workshop from Software Requirements to Architectures (STRAW '03), Portland, USA, pp. 69–76.

Liu, L., Yu, Eric, 2003. From requirements to architectural design – using goals and scenarios. In: 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. In: 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. In: 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. In: 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. In: 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. 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. In: EASE'03 – Proceedings 7th

International Conference on Empirical Assessment & Evaluation in Software Engineering, Keel, UK.

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.

Sheskin, D.J., 2004. Handbook of Parametric and Non-parametric 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, eigth ed. Addison Wesley.

Thelin, Thomas, 2004. Team-based fault content estimation in the software inspection process. In: 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 quantitative study. Journal of Systems and Software (January), 1–18.

Wang, Z., Sherdil, K., Madhavji, N.H., 2005. ACCA: an architecture-centric concern analysis method. In: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, USA, pp. 99–108.

WebStat, 2000. School of Psychology, New England University. Available from: <http://www.une.edu.au/WebStat/unit_materials/c5_inferential_statistics/what_alpha_level.html>.

(WICSA '07) 2007. Working IEEE/IFIP Conference on Software Architecture, Mumbai, 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.

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

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 29 (4), 315–321.

Remo Ferrari is a doctoral candidate at the University of Western Ontario, London, Ontario, Canada. His research interest is in Software Engineering, specifically in the areas of Software Architecture, Requirements and Project Management. In particular, his work has investigated these areas through an empirical viewpoint, examining such issues as the technical effects an architecture has on new requirements, and the impact of human agents background experience on software architecting. His primary research goal is to help in advancing the underlying scientific knowledge and theory with respect to these and related issues. This goal is being pursued in two phases: to first conduct "laboratory" or exploratory studies to generate preliminary results and then, based on these findings, to conduct industrial studies. Such empirical work is intended to form underlying grounded theory on which processes, methods, tools, etc. can be developed. In addition to research, he teaches both Software Engineering and Computer Science courses.

Nazim H. Madhavji is a Professor in the Department of Computer Science at the University of Western Ontario, Canada. He obtained his Ph.D. from the University of Manchester, England, in 1980. His research interests includes: software requirements; software architectures; evolution of software; software quality and measurements; defect tracking and analysis; congruence between software products and processes; and empirical studies.

He has led a number of research projects in software engineering, involving corporations such as IBM Canada, DMR Group, CAE Electronics, Transport Canada, and CRIM, and was a Principal Investigator in several multi-university projects. He is the chief architect and editor of the 27-chapter book "Software Evolution and Feedback: Theory and Practice" with Juan F. Ramil and Dewayne Perry, John Wiley, 2006. He is an Editor (with Khaled El Emam) of the book: "Elements of Software Process Assessment and Improvement", IEEE Computer Society Press, 1999. He is on the Editorial Boards of several scientific journals. He is a consultant to several organisations in the field of software and is a consultant to several universities internationally in the areas of Software Engineering research, pedagogy, and faculty development.