Identifying and Understanding Architectural Risks in Software Evolution: An Empirical Study

Odd Petter Nord Slyngstad[1], Jingyue Li[1], Reidar Conradi[1], M. Ali Babar[2]

[1] Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU), Trondheim, Norway
{oslyngst, jingyue, conradi}@idi.ntnu.no
[2] LERO– The Irish Software Engineering Centre, University of Limerick, Limerick, Ireland
malibaba@lero.ie

Abstract. *Software risk management studies commonly focus on project level risks and strategies. Software architecture investigations are often concerned with the design, implementation and maintenance of the architecture. However, there has been little effort to study risk management in the context of software architecture. We have identified risks and corresponding management strategies specific to software architecture evolution as they occur in industry, from interviews with 16 Norwegian IT-professionals. The most influential (and frequent) risk was "Lack of stakeholder communication affected implementation of new and changed architectural requirements negatively". The second most frequent risk was "Poor clustering of functionality affected performance negatively". Architects focus mainly on architecture creation. However, their awareness of needed improvements in architecture evaluation and documentation is increasing. Most have no formally defined/documented architecture evaluation method, nor mention it as a mitigation strategy. Instead, problems are fixed as they occur, e.g. to obtain the missing artefacts.*

Keywords: software architecture, software evolution, risk management, software architecture evaluation

1 Introduction

Modern software systems are commonly built by acquiring and integrating various components developed by commercial or open source entities. The software engineering community has enabled several processes for developing and maintaining component-based systems. Proper handling of software architecture is one of the most important factors towards successful development and evolution of component-based systems. However, there has been little effort to identify and understand the architectural risks in software evolution and potential strategies to deal with those risks. We assert that it is important to obtain and disseminate the information about potential risks (i.e. problems) in architecture evolution, as the architecture constitutes the central part of a software system [1]. Knowledge and understanding about architecture evolution risks should facilitate the development of improved strategies to mitigate these risks.

We have decided to obtain such knowledge from practicing IT-professionals working with software architecture, as they are expected to encounter risks (i.e. problems that may occur) in evolving software architectures on a regular basis. Our research here is concerned with Component-Based Software Engineering (CBSE) development, where there has been architectural evolution during the systems' lifetime.

Using a convenience sample of respondents, we carry out a preliminary investigation of architectural risks and management strategies in software evolution. This means changes to the structure(s) of a system of software elements, their external properties and mutual relationships, all viewed from a perspective of risk analysis and risk mitigation. This exploratory study is targeted at Norwegian IT-professionals who hold significant knowledge and experience in designing and evolving software architectures.

We have identified architectural risks (i.e. problems identified in planning or experienced during the maintenance/evolution) and associated risk management strategies (i.e. methods to mitigate these issues) as they occur in industry. "Lack of stakeholder communication affected implementation of new and changed architectural requirements negatively" was the most influential as well as the most frequent risk. This risk was most effectively mitigated by extending the time used towards communication with stakeholders. "Poor clustering of functionality affected performance negatively" was the next most frequent risk. This risk was in turn most successfully mitigated by refactoring or improving the modifiability of the architecture.

Furthermore, architects easily handle anticipated or experienced risks. However, their focus is usually on "forward engineering", not on reengineering (i.e. the architecture solution rather than the suitable steps to get there [14] in advance). Despite this, some of the findings also show that awareness of software documentation and evaluation issues and practices is increasing. Also, most of the respondents have no formally defined or documented architecture evaluation method in place. Rather, challenges are met as they appear, and the main focus is on obtaining the missing artifact. Finally, none of our respondents mentioned using formally defined or documented architecture evaluation as a risk mitigation strategy.

The remainder of the paper is organized as follows: Section 2 holds Background. Research Design is in Section 3. Section 4 contains information on our data collection, and the results of our study are in Section 5. Discussion and Threats to Validity are located in Section 6, and Conclusions and future work are in section 7.

2 Background and Related Work

Software Architecture [1] can be defined as the discipline dealing with the structure or structures of a system, comprising software elements, the externally visible properties ("interface" of in-going and out-going calls) of those elements, and the relationships between them. Well-defined software architecture is one of the key factors in successfully developing and evolving a non-trivial system or a family of systems. A well-defined software architecture provides a framework for the earliest design decisions to achieve functional and quality requirements. In addition, it has a profound influence on project decomposition and coordination. Poor architecture often leads to project inefficiencies, poor communication, and inaccurate decision making [1]. The above definition of software architecture refers to software elements, which can be seen as components of the given software system. Hence software architecture is closely related to CBSE [2].

Clerc et. al. [14] conducted a study to understand architects' attitudes towards software architecture knowledge. They found that architects are aiming more at creation and communication instead of review and maintenance of a system's architecture. Bass et al. [21] analyzed the output from 18 ATAM evaluations to discover risk themes specifically for software architecture. Besides a set of risk categories, they found that the more prevalent risks are those of omission (i.e. of <u>not</u> taking action on a particular issue). They also did <u>not</u> find a link between the risk categories and the business/mission goals or the domain of a system. Bass et al. further comment that the similarities to their study shown in [23] indicate the industrial relevance of the risk categories [21], as well as the ability of ATAM analysis to discover architectural risks. Another risk categorization from ATAM evaluations is presented in O'Connell [22], using 8 evaluation results. Although study [22] was analyzed independently from [21], the resulting themes are similar in content. It should be noted though that neither of these studies deal explicitly with the <u>evolution</u> of software architecture.

The architecture of a system will evolve as architectural changes are accumulated over time. There are diverging views in the research community about how software evolution should be defined. These include considering maintenance as a broader term [5], seeing evolution as a step in the software lifecycle [4], and regarding evolution as software systems' dynamic behavior through maintenance and enhancements [3]. Some [9] consider evolution as the enhancement and improvement performed on a system between releases. Based on this description, we define software evolution for this study as: *the systematic and dynamic updating in new/current development or reengineering from past development of component(s) (source code) or other artifact(s) to <u>a)</u> accommodate new functionality, <u>b)</u> improve the existing functionality, or <u>c)</u>enhance the performance or other quality attribute(s) of such artifact(s) between different releases.*

If left unchecked, over time, a system's architecture will naturally decay as new quality and functional requirements are imposed on it. This decay is manifested by the original architectural structure(s) being lost. This is sometimes called "software rot" [20], and is one of the most prevalent reasons behind reengineering the architecture of a software system.

Risk management entails methods to mitigate risks that may occur during a software development project. Boehm [8] describes a framework for risk management consisting of two main steps, namely risk assessment (identification, analysis, and prioritization) and risk control (planning, resolution, and monitoring). Ropponen and Lyytinen [6] have identified six elements of software risk. Their results reveal influence on risk elements by environmental factors (e.g. development method). Also, awareness of risk management importance and method(s) was shown to have an effect. Keil et al. [10] conducted a risk management survey of project managers. They identified several additional important risk factors in comparison with Boehm

[8], contributing these to changes in the industry since Boehm’s study. Additionally, they discovered that important risks were commonly out of managers’ control. They therefore suggested that project managers widen their attention beyond traditional software risk factors.

Further based on the definition of risk in Boehm’s article[8], as well as input from [6][12], we use the following definition for architectural evolution risks: *the issues or problems that can potentially have negative effects on the software architecture of a system as it evolves over time, hence compromising the continued success of the architecture*. The above studies on architectural risks [21][22] have focused on discovering risk categories directly from the output of ATAM [1] analyses. They use analysis outputs from organizations where such evaluation is an established practice. However, they do not comment on how commonly such formal evaluation methods are used in industry. Nor do they take software evolution specifically into account. In [7], the authors found that evaluation practices could range from completely ad-hoc to formally planned, from qualitative to quantitative. They also discovered that the approach depended on the goals of the evaluation. This means that additional risk issues and management strategies could be left undiscovered by looking only at output from structured analysis reports. We therefore decided to employ semi-structured interviews to gather qualitative information on risk issues and risk management strategies.

3 Research Design: Context, Motivation and Research Questions

We observed that risks and risk management strategies are commonly studied in relation to general software development [11][12][13], identifying risks on the project level [6][8][10]. Similarly, software architecture studies often focus on the design, implementation and maintenance of the architecture. While these results are important, there has been little effort to study risk management in the context of software architecture [21][22]. Hence, we decided to carry out an empirical study to help further identify and better understand the risks and risk management strategies in relation to software architecture.

This research is limited to those software systems which have two major characteristics: use of CBSE and changes in the systems' software architectures during their lifetimes. This means projects that have at least delivered the first production release, i.e. can be said to be in the “maintenance” phase.

Our main motivation is to obtain insight into the actual risks (i.e. issues identified and experienced which may affect the software architecture negatively) and associated risk management strategies (i.e. effective mitigation methods), as they occur in industry, in relation to software architecture evolution. We aim to use the results from this exploratory study as basis for more in-depth studies in this area.

This study is aimed at identifying and understanding risks and strategies relevant to software architecture evolution. That is, we investigate the steps of risk identification, analysis and prioritization, as well as risk planning and resolution [8], as they occur in industry. We do not cover issues pertaining to risk assurance or monitoring [8]. The research questions are as follows:

RQ1: What are the relevant architectural risks of software evolution, i.e. what software architecture related risks can be encountered during software evolution? Any issue that can affect a project adversely if not handled correctly is considered a risk [8]. The first step in Boehm's risk management framework [8] entails risk identification, analysis, and prioritization. We are hence here interested in investigating the state-of-the-practice regarding risk awareness, i.e. to obtain insight on which risks that software architects deem more important in relation to software architecture evolution.

As aforementioned, software architecture is the central part of a software system [1], so failure of the software architecture can easily cause the entire project to fail. Hence a proper focus on the software architecture is needed to ensure the project is kept on budget and schedule. Similarly, changes to the software architecture can cause subsequent changes in many components of a software system [1]. It is therefore imperative to be aware of the possible risks incurred on the software architecture through software evolution.

RQ2: How can these risks best be assessed; through which methods or mechanisms were these risks identified, analyzed and prioritized? Software architecture evaluation is widely known as an important and effective way to assess architectural risks [1, 7]. In order to identify, analyze and prioritize [8] risks there is the need for effective methods or mechanisms for software architecture evaluation. Such mechanisms help validate architecture design decisions with respect to required quality attributes (such as testability, availability, modifiability, performance, usability, security etc.). Prior architecture analysis studies [21][22] focused on structured analysis outputs as a method to discover risks. However the analysis methods used can range quite widely [7]. Investigating a wider range of analysis methods will help discover risk issues possibly missed by earlier studies.

RQ3: How can these risks best be mitigated: what were the relevant risk management strategies? Were the strategies successful or not? The second step in Boehm's framework [8] encompasses risk control. This step focuses on problem mitigation; it is aimed at handling problems to minimize their impact. Here, our aim is to obtain the status quo, and suggest possible improvements by enabling a systematic approach to architectural risk management in software evolution. It is therefore imperative that we receive information on both positive and negative aspects of employed risk management strategies, and also on their outcomes.

Again, risks in relation to the central part of a software system (i.e. the architecture [1]) are important. Proper management of these risks on the three levels, technical, process and organization [11][12][13], provides the ability to minimize the potentially far-reaching impacts of these risks [8].

In order to practically explore the three research questions above, we designed an interview guide consisting of six questions. The relation between the questions in the interview guide, the research questions, and Boehm's framework [8] is shown in Table 1.

Table 1. Relation between research questions and the interview guide

	Identification, Analysis, and Prioritization [8]	**Assessment [8]**	**Planning, and Resolution [8]**

Questions in the interview guide	**RQ1**	**RQ2**	**RQ3**
Q1.1. Describe architectural problems (indicate influence) and strategies (rate outcome) you **identified in planning** maintenance/evolution**?**	X		X
Q1.2. Describe architectural problems (indicate influence) and strategies (rate outcome) **experienced and employed during** maintenance/evolution?	X		X
Q2. Indicate weighting of and any changes in the following quality attributes[1]: testability, availability, modifiability, performance, usability and security) in your software architecture?	X		
Q3. How has the architecture changed throughout the lifetime of the system?	X		
Q4. Please describe your architecture change process?		X	X
Q5 Which architectural patterns (e.g. layering, task control, AI approach pipe-and-filter etc.) did you use to design the architecture?	X		
Q6. Does your organization use a defined and/or documented method or process to evaluate software architecture?		X	X

Question Q6 has been adapted from an earlier empirical study aimed at identifying the factors that can influence software architecture evaluation practices [7]. We also gathered demographic data (e.g. level of experience) about the respondents. The interview guide was piloted with 3 researchers to ensure quality and ease of understanding, through which the questions were polished and refined. We aimed to be flexible so as to gain as much qualitative information on each question as possible. Therefore, all the questions (Q1-Q6) were left open-ended. Also, the influence of each risk and the outcome of each strategy were indicated on a 5-point Likert scale. That is, risk Influence was ranked Very High = 5 to Very Low = 1. Similarly, strategy Outcome success was ranked Completely = 5, Mostly = 4, Medium = 3, Somewhat = 2 and Not at all = 1 successful.

4 Data Collection and Analysis

This study was carried out using a convenience sample of participants from the software industry in Norway. Potential respondents were first contacted by email, and sent the invitation letter with interview guide to get an overview. Later the potential respondents were contacted again by phone and signed up for a phone-interview appointment if they agreed to participate. The respondents were 16 IT-professionals in different companies with prior knowledge and experience with software architecture.

The phone interviews took on average 30 minutes to carry out, and we obtained complete responses to all the six questions from all 16 respondents. The data was recorded on paper and transcribed into electronic form. The responses were also summarized and read back to the respondents directly after the interviews, so they could be checked for accuracy.

Nine of the respondents had bachelor level degrees, while seven had master degree level educations. On average, the respondents had 8 years of experience working with software architecture, with six having less than five years of experience, five having 5-10 years of experience and another five having over 10 years of experience

We analyzed the data as follows: The data was initially analyzed by dividing the data into discrete parts and coding each piece according to risk or strategy theme(s). As an example, for risks this was done as {condition – what may go wrong, consequence(s)}: e.g. “requirements from earlier versions still in effect affected architecture design negatively.” was coded as {earlier version requirements, negative for architecture design}.

We then examined them for commonalities and differences, and grouped related pieces of information based on their coding (e.g. for risks, {earlier version requirements, negative for architecture design} and {required same functionality as before, negative for planning} were grouped as {required backward compatibility, negative for architecture maintenance/evolution planning and design}). Each respondent’s transcript was run through this procedure. The results were checked by a second researcher to ensure reliability. This is similar to the constant comparison method described in [16]. The issues identified in the data analysis were classified into three categories; technical, process and organizational. We believe that risk management is not merely a technical issue; rather, it spans all three categories [11][12][13][21].

5 Results

The results are here divided into categories of (1) technical, (2) process and (3) organizational risks. This means that we have combined the findings from Q1.1 and Q1.2 for RQ1 and RQ3.

Technical risks: Table 2 shows the most influential technical risks and corresponding management strategies performed. From Table 2, we can see that the strategy applied in planning towards TR1 was Completely successful (Outcome = 5). Furthermore, overall the strategies were also 3 out of 5 of Medium success (Outcome = 3), and 1 out of 5 Not at all successful (Outcome = 1).

Table 2. Most **influential (Influence ≥ 4) technical** risks (TRs) and corresponding management strategies performed

Technical	ID	Risk	Influence	Strategy	Outcome
Identified in planning	TR 1	Poor clustering of functionality affected performance negatively	4	Refactoring of the architecture	5
Experienced during	TR 2	Poor original core design prolonged the duration of the maintenance/ evolution cycle	4	Improve modifiability of the architecture	3
	TR 3	Increased focus on modifiability contributed	4	Implementation of changes towards	3

		negatively towards system performance		modifiability	
	TR 4	Varying release cycles for COTS/OSS components made it difficult to implement required changes	4	Use own development as potential backup	3
	TR 5	Poor clustering of functionality affected the performance negatively	4	Implement extra architecture add-ons	1

Process risks: Table 3 (below) shows the most influential process risks and corresponding management strategies performed. These results (Table 3) show that all of the strategies used in response to the most influential risks in planning were Completely successful. Towards the risks experienced during the maintenance/evolution, the strategies were 3 out of 10 Completely successful, 5 out of 10 of Medium success, while 2 out of 10 were Completely successful.

Table 3. Most **influential (influence ≥ 4) process** risks (PRs) and corresponding management strategies performed

Process	ID	Risk	Influence	Strategy	Outcome
Identified in planning	PR1	Lack of architecture documentation contributed to more effort being used on planning the maintenance/ evolution	4	Recover arch. documentation from current architecture design	5
	PR2	Lack of architecture evaluation delayed important maintenance/ evolution decisions	4	Recover evaluation artefacts where needed	5
				Alter process to capture important details	5
Experienced during	PR3	Lack of stakeholder communication affected implementation of new/ changed architectural requirements negatively	5	Negotiated project extension	3
				Allow additional time for communication/feedback	5
	PR4	Insufficient requirements negotiation contributed to requirement incompatibilities on the architecture	4	Postponed some requirements to next maintenance/evolution cycle	3
	PR5	Poor integration of architecture changes into implementation process affected implementation process and the architecture design negatively	4	Overlay new architecture change process onto implementation process	5
				Integrate architecture considerations into implementation process	3
	PR6	Using Software Change Management (SCM) sys. w/o explicit software architecture description contributed to inaccuracies in communicating the	4	Use separate system for architecture description (using ADL), link to SCM system	3
				Trial use of additional ADL system	3

		architecture			
	PR7	No standard terminology affected internal and external communication efforts negatively	4	Align terminology with literature	1
				Extra communication to clarify terminology	1
	PR8	Customer architects being unfamiliar with architecture change process affected maint./ evo. cycle schedule negatively	4	Extra communication effort with own resident architect to clarify	5

Organizational risks: Table 4 (below) shows the most influential organizational risks and corresponding management strategies performed. Among the strategies used in response to these most influential organizational risks (Table 4) identified in planning, 2 out of 4 were Medium successful, while 2 out of 4 were Completely successful. Towards those experienced during, the strategies were all Medium successful.

Table 4. Most **influential (influence ≥ 4) organizational** risks (ORs) and corresponding management strategies performed

Organization	ID	Risk	Influence	Strategy	Outcome
Identified in planning	OR 1	Architecture team on a per maintenance/evolution cycle basis contributed to loss of knowledge about the existing architectural design	4	Dedicated personnel to “retrieve” knowledge	3
	OR 2	Cooperative maintenance / evolution with architects from customer organization required extra training and communication efforts	4	Frequent, interactive, scheduled meetings to keep up to date	5
	OR 3	Lack of clear point of contact from customer organization contributed to inconsistencies in communication of the architecture and requirements	4	Involve all “layers” of customer organization as stakeholders, allow extra communication time	5
	OR 4	Not allowed to change OSS as decision mandate external to architecture team, affecting performance and modifiability negatively	4	Ensure compliance with external mandate holder	3
Experienced during	OR 5	Separate architecture team per maint. / evo. cycle contributed to insufficient knowledge about the existing architectural design	4	Regain architecture details from upper management remaining	3
	OR 6	Prior architecture maint./ evo. by other projects due to lack of personnel made it	4	Merge architecture knowledge / documentation to	3

		difficult to obtain existing architecture design documentation		central location	
	OR 7	Large architecture team affected division of duties and subsequently implementation of maint./ evolution cycle negatively	4	Divide duties between subgroups	3
	OR 8	Lack of clear lead architect affected implementation progress negatively and contributed to extra effort needed	4	Merge duties and diverge roles more clearly	3

Additionally, our results show that the overall most frequent (and most influential) risk was “Lack of stakeholder communication affected implementation of new and changed architectural requirements negatively”. The most successful strategy in response to this risk was “Allow additional time for communication for communication and feedback”. The second most frequent risk was “Poor clustering of functionality affected performance negatively”, with “Refactoring the architecture” and “Improve the modifiability of the architecture” as corresponding most successful strategies. The results from questions Q2, Q3, Q5 (Table 5), and Q4, Q6 are below.

Table 5. Summary of additional findings for RQ1

Q2. Quality attribute foci:	
• Focus on any given QA can change during the project. • Only a few projects experienced a lowering of focus on a given QA. • Most frequent QA with increased focus was Modifiability, followed by Usability.	
Q3. Architecture changes made during system lifetime to:	
• Improve processing speed or scale (7 out of 16) • Improve flexibility to accommodate future changes (7 out of 16) • Accommodate new or altered user requirements (5 out of 16)	• Improve system uptime (3 out of 16) • Enable additional access interfaces (1 out of 16) • Increase abstraction level (1 out of 16) • Support additional record types (1 out of 16)
Q5. Architectural patterns used (as means to solve design challenges):	
• Inversion of Control (1 out of 16), • Layered (3 out of 16), • Blackboard (3 out of 16),	• Model View Controller (4 out of 16), • Pipeline (3 out of 16), • Task Control (2 out of 16), and • Broker (1 out of 16).

The following are results from Q4 (**RQ2, RQ3**) (architecture change process):

- none used a strictly defined change process,
- 7 out of 16 performed this process informally,
- 4 out of 16 employed loosely defined procedures,
- 3 out of 16 changed the architecture as part of the development process, and
- 2 out of 16 just change the architecture as needed.

In question Q6, none of the respondents answered that they have a defined or documented process for software architecture evaluation. 5 out of 16 of the respondents have a loosely defined process in place if needed. Another 5 out of 16 have knowledge of evaluation processes or methods mentioned in literature. Yet another 5 out of 16 of the respondents carry out a software architecture evaluation informally if needed. Finally, 1 out of 16 of the respondents reports that her/his organization has a process for software architecture evaluation in place (in this specific case, based on the Architecture Tradeoff Analysis Method – ATAM [1]), but this is not commonly used.

6 Discussion

6.1 Comparison to related work

The Technical risks identified by the respondents show a high focus on design and creation of the architecture, supporting [14].

While Ropponen's [6] focus was overall software development risks, ours is software architecture risks in software evolution. The strategies used in response to the risks we identified as (See Table 6 below) "Architecture Team" and "Requirements" risks were reported as being Medium or Completely successful in outcome. We can hence support the notion that there is at least some success in managing risks related to "Architecture Team" and "Requirements" [6].

A summarized comparison with the above and Bass et al. [21] is also in Table 6.

Table 6. Summary of comparison to related work

ID	**Ropponen et al. [6]**
	Requirements risks:
PR4	"Insufficient requirements negotiation contributed to requirement incompatibilities"
TR3	"Increased focus on modifiability contributed negatively towards system performance"
	Architecture Team risks:
OR5	"Separate architecture team per maint. / evo. cycle contributed to insufficient knowledge about the existing architectural design"
OR7	"Large architecture team affected division of duties and subsequently implementation of maint./ evo. cycle negatively"
OR8	"Lack of clear lead architect affected implementation progress negatively and contributed to extra effort needed"
	Stakeholder risks (from the subcontractor viewpoint):
PR3	"Lack of stakeholder communication affected implementation of maint./ evo. cycle negatively"
OR2	"Cooperative maint./evo. w/ architects from customer organization required extra training and communication efforts"
OR3	"Lack of clear point of contact from customer organization contributed to inconsistencies in communication of the architecture and requirements"
PR8	"Customer architects being unfamiliar with architecture change process affected maint./evo cycle schedule negatively"
ID	**Bass et al. [21]**
	Quality Attribute risk:
TR3	"Increased focus on modifiability contributed negatively towards system performance"
	Integration risks:
TR4	"Varying release cycles for COTS/OSS components made it difficult to implement required changes"
OR4	"Not allowed to change OSS as decision mandate external to architecture team, affecting performance and modifiability negatively"

	Requirements risks:
PR4	“Insufficient requirements negotiation contributed to requirement incompatibilities on the architecture”
TR3	“Increased focus on modifiability contributed negatively towards system performance”
	Documentation risks:
PR1	“Lack of architecture documentation contributed to more effort being used on planning the maintenance/evolution”
PR6	“Using Software Change Management system w/o explicit software architecture description contributed to inaccuracies in communicating the architecture”
	Process and Tools risks:
PR2	“Lack of architecture evaluation delayed important maintenance/evolution decisions”
PR6	“Using Software Change Management system w/o explicit software architecture description contributed to inaccuracies in communicating the architecture”
	Allocation risks:
TR1	“Poor clustering of functionality affected performance negatively”
TR4	“Varying release cycles for COTS/OSS components made it difficult to implement required changes”
	Coordination risks:
PR3	“Lack of stakeholder communication affected implementation of maint./evo. cycle negatively”
PR8	“Customer architects being unfamiliar with architecture change process affected maint./evo cycle schedule negatively”
OR2	“Cooperative maint./evo. with architects from customer organization required extra training and communication efforts”
OR3	“Lack of clear point of contact from customer organization contributed to inconsistencies in communication of the architecture and requirements”
OR4	“Not allowed to change OSS as decision mandate external to architecture team, affecting performance and modifiability negatively”

6.2 Observations on key architectural risks and promising risk management strategies

The most influential Process risks we identified (Table 3) show that the **main focus is still forward thinking (producing systems according to budget and schedule) rather than hindsight reflection and learning.** Further, from the answers to Q5 we can see that the consequences of using one or more specific patterns are neither explicitly considered, nor evaluated as potential risks (though tactics, packaged by patterns, is a risk issue also discovered from ATAM reports in [21][22]).

The answers from Q4 and Q6 also point towards this main focus. Hence there is no apparent specific focus on discovering potential problems (rather problems are fixed as they are encountered, focussing on the missing artefacts). This is despite the potential benefits (e.g. identifying architecture design errors and potentially conflicting quality requirements early) of defined and documented architecture evaluation described in the literature [1]. However, architects are **becoming aware that their practices around evaluation and documentation need improvement.** This is echoed by the Organizational risks we identified (Table 4), such as “Architecture team on a per maint./evo. cycle basis contributed to loss of knowledge about the existing architectural design” and “Large architecture team affected division of duties and subsequently implementation of maint./evo. cycle negatively”.

A **link to Business Risks** [19] (i.e. those that affect the viability of a software system) can also be seen. The architectural risks identified are influenced by and in turn also affect such elements as e.g. cost, schedule.

Considering the most influential **Technical risks** (table 3), we can see that the majority of them were experienced during the maintenance/evolution, without prior planning. The same appears the case for the most influential **Process risks,** whereas for the most influential **Organizational risks** half were identified in planning, and another half were experienced during the maintenance/evolution. In terms of management strategies, one overall trend appears to be that those employed in response to risks identified in planning had a more successful outcome. This appears especially to be the case where the same risk was both identified in planning as well as experienced during the maintenance/evolution (e.g. Technical risks TR1 and TR5: "Poor clustering of functionality affected performance negatively"). These findings also emphasize the points about forward engineering and awareness discussed above.

One of the strategies applied towards technical risks, as well as two of the strategies applied towards process risks were Not at all successful. These strategies should be viewed in light of the respective projects' context (Tables 2, 3). Additionally, improvement is needed in the employed strategies, especially regarding issues encountered during maintenance/evolution. The lack of a strictly defined and documented architecture change process reported by the respondents (Q4) is also an interesting finding. We would expect architecture evaluation to be part of a given change process in order to analyze the consequences of proposed architectural changes.

To improve this situation, we believe that rigorous documentation and evaluation of architecture should be made an integral part of a software architecture change process. Furthermore, management of risks specific to architectural modifications should be given more attention. To achieve these objectives, software architects should be provided appropriate training. Moreover, organizational management should also demonstrate commitment to implement changes to the way software architecture changes are handled.

6.3 Threats to validity

Threats to validity (using definitions provided by Wohlin et al. [15]):

Construct Validity: The research questions are rooted firmly in the research literature, and the actual questions in the interview guide have direct relations to the research questions. The interview guide was refined through pre-testing among our colleagues to ensure quality. All the terms used in the guide were defined at the beginning to avoid any potential misinterpretations.

External Validity: This study has been conducted by using a convenience sample of 16 IT-professionals, an issue which remains a threat. Nevertheless, obtaining a random sample is almost unachievable in software engineering studies because our community lacks good demographic information about populations of interest [17]. The respondents were chosen by us based on their background and experience with software architecture. Each respondent nevertheless represents a different company.

Internal Validity: The respondents are all knowledgeable and from the software industry, and have expressed an interest in the study. They all have the needed knowledge and background to provide informed answers. We hence believe that they have answered the questions to the best of their ability, truthfully and honestly, drawing on their own experiences, skills and knowledge. We also clarified any ambiguities in the questions or the accompanying definitions during the actual interviews, in addition to the definitions provided in the guide.

Conclusion Validity: This is an exploratory study. The findings are based on analyzing data from a relatively small number of software architects. We plan to implement a large scale study to confirm the results of this study. However, the exploratory nature of the study has identified several issues that may cause architectural risks for evolving systems. The insights gained will also function as background for refining the interview guide towards expansion of the sampling base for the planned larger scale study.

7 Conclusion and Future Work

We conducted phone-based, semi-structured interviews of 16 software architects from Norway for an exploratory study regarding risks and risk management strategies occurring in industry related to software architecture evolution.

Our findings include an initial identification of risks and corresponding risk management strategies as they occur in industry. Our main observations include that "lack of stakeholder communication affected implementation of new and changed architectural requirements negatively" was the most influential and frequent risk. The corresponding most successful strategy was to "Allow additional time for communication and feedback". In second place concerning most frequent risks came "Poor clustering of functionality affected performance negatively". The most successful management strategies towards this risk were "Refactoring the architecture", and "Improve the modifiability of the architecture".

Furthermore, architects' main concerns are towards designing and creating the architecture. However, our results also show some awareness towards improvements in relation to how these tasks are performed, as well as towards the importance of retaining knowledge about and performing evaluation of the architecture. As most respondents have no formally defined or documented method to evaluate software architecture, problems are fixed as they occur with focus on the lacking artefacts rather than on the method.

Our results here will be used as input for a larger study in the software industry to survey the state-of-practice on risk and risk management regarding software architecture evolution. In particular, we plan to explore the relation between risks and risk management practices, and project context factors.

Acknowledgements

We thank all parties involved. The study was performed in the SEVO project, a Norwegian R&D project in 2004-2008 with contract number 159916/V30.

References

1. L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice,* Second Edition, Addison-Wesley, 2004.
2. L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, K. Wallnau, *Volume I: Market Assessment of Component-based Software Engineering* in SEI Technical Report number CMU/SEI-2001-TN-007, 2001.
3. L. A. Belady and M. M. Lehman; *A model of a Large Program Development,* IBM Systems Journal, 15(1):225-252, 1976.
4. K. H. Bennett and V. Rajlich; *Software Maintenance and Evolution: A Roadmap,* ICSE'2000 – Future of Software Engineering, Limerick, Ireland, pp. 73-87, 2000.
5. I. Sommerville; *Software Engineering,* Sixth Edition, Addison-Wesley, 728 p., 2001.
6. J. Ropponen and K. Lyytinen, *Components of Software Development Risk: How to Address Them? A Project Manager Survey*, IEEE Transactions on Software Engineering, 26(2), 98-112, Feb. 2000.
7. M. Ali Babar, L. Bass, I. Gorton, *Factors Influencing Industrial Practices of Software Architecture Evaluation: An Empirical Investigation*, QoSA 2007, Medford, Massachusetts, USA, July 12-13, 2007.
8. B. W. Boehm, *Software Risk management: Principles and Practices,* IEEE Software, 8(1), 32-41, January 1991.
9. M. Carr, S. Kondra, I. Monarch, F. Ulrich, and C. Walker, *Taxonomy-Based Risk Identification*, Technical Report SEI-93-TR-006, SEI, Pittsburgh, USA, 1993.
10. M. Keil, P. E. Kule, K. Lyytinen and R. C. Schmidt, *A Framework for Identifying Softare Project Risks,* Communications of the ACM, 4(11), 76-83, November 1998.
11. B. W. Boehm, *A Spiral Model of Software Development and Enhancement,* IEEE Computer, 21(5), 61-72, May 1988.
12. A. Gemmer, *Risk Management: Moving Beyond Process,* IEEE Computer, 30(5), 33-41, May 1997.
13. H. Hecht, *Systems Reliability and Failure Prevention,* Artech House Publishers, 2004.
14. V. Clerc, P. Lago, H. van Vliet, *The Architect's Mindset*, QoSA 2007, Medford, Massachusetts, USA, July 12-13, 2007.
15. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering – An Introduction,* Kluwer Academic Publishers, 2002.
16. C. B. Seaman, *Qualitative Methods in Empirical Studies of Software Engineering,* IEEE Transactions on Software Engineering 25(4):557-572, July/August 1999.
17. T. C. Lethbridge, S. E. Sim, and J. Singer, Studying Software Engineers: Data Collection Techniques for Software Field Studies, Empirical Software Engineering, 10(3), 311-341, July 2005.
18. B. Kitchenham and S. L. Pfleeger, *Principles of Survey Research, Parts 1 to 6,* ACM Software Engineering Notes, 2001 – 2002.
19. D. G. Messerschmitt and C. Szyperski, *Marketplace Issues in Software Planning and Design, IEEE Software* **21** (3): 62–70, May/June 2004.
20. R. E. Johnson and B. Foote. "Designing Reusable Classes." Journal of Object-Oriented Programming, 1(2):22-35, 1988.
21. L. Bass, R. Nord, W. Wood, D. Zubrow, *Risk Themes Discovered Through Architecture Evaluations,* Proc. WICSA 2007
22. O'Connell, D. "Boeing's Experiences using the SEI ATAM® and QAW Processes", April, 2006, http://www.sei.cmu.edu/architecture/saturn/2006/OConnell.pdf
23. Charette, R.N., Why software fails, Spectrum, IEEE Volume 42, Issue 9, Sept. 2005 Page(s):42 – 49.