Assessing the architectonics of large, software-intensive systems using a knowledge-based approach

Christian Del Rosso
Nokia Research Center
P.O Box 407, 00045, Finland
christian.del-rosso@nokia.com

Alessandro Maccari
Nokia Networks
Via Bombay 5, 00144, ROMA, Italy
alessandro.maccari@nokia.com

Abstract

This paper presents an assessment case study on the evolutionary capability of a large software system using a knowledge-based approach. The knowledge-based assessment is based on interviews with selected stakeholders of a software system. We have used this to assess the capability of the software architecture to evolve in one large Nokia software system. We have found that this approach proves to be effective in large organizations where development teams are distributed in different time-zones, with cultural differences and with limited person-to person communication. The lessons learned and the advantages of using this approach are presented.

1 Introduction

The architectonics of a software system [18] is defined as the categoric differentiation of layers according to any or some of the following attributes: stability, constraining power, ease of change, likelihood of change, scale of effect, magnitude of change. Architectonics is all about layering the architecture of the system in order to group architecturally significant entities into categories that help the analysis of system-wide properties.

For instance, a typical architectonics view groups architecturally significant entities according to their ease of change. As it happens with buildings, software systems that have a clear separation between architectural elements according to their capability to evolve usually last for a longer time, and are less expensive to maintain. In a business context where development costs usually are a fraction of maintenance costs [11, 21] (when considered throughout the whole system lifetime), having a clear picture of the evolvability

of system entities can help architects make educated decisions whenever new requirements are examined for implementation.

To facilitate evolution, architectural assessments have been conceived as a way to evaluate significant architectural (i.e. system-wide) properties of a software system. For instance, a lift management system may be assessed in order to determine whether it fulfills certain reliability requirements; an instant messaging software platform may be assessed in order to figure out its performance in the busy hour (load assessment); an off-the-shelf software component may be assessed in order to figure out whether the API that it exposes is easy to integrate with another system; and so on.

Software architecture assessments can be classified according to several criteria. A first classification can distinguish the assessments in two broad categories:

- Functional assessments. These aim to evaluate the functional properties of a system. For instance, a functional assessment could be carried out in order to verify whether a system's web interface supports secure HTTP and Japanese language.

- Qualitative assessments. These assessments have the goal to evaluate the qualitative properties of a certain system. For instance, a qualitative assessment could be carried out in order to verify whether a mission-critical transmission system is highly available.

Assessments can also be classified according to the sort of information that is used to carry out the assessment. Software architecture documentation and description influence the choice of the particular software architecture assessment method used. Some sort of software architecture description is used as input by all software assessment methods.

From the standard IEEE 1471 [13] follows that every software system has an architecture. Furthermore,

a software architecture and a software architecture description are not the same thing. The architecture is implicitly or explicitly described according to standards or to informal notations. In some cases, the software architecture may not be described at all.

Current scientific literature abounds with examples of architecture description languages and methods to describe software architecture. For instance, the Unified Modeling Language (UML) [25] is one of the best-known modeling and description languages for software architectures. A whole book by Clements et al. [3] focuses entirely on documenting software architectures. An in depth review of different software architecture description languages is given in [19].

Model-based assessments are based on a model of the system. For instance, a UML diagram of a software system could be the basis of a model-based assessment. The model-driven architecture movement (MDA) in the Object Management Group (OMG) addresses software design and evaluation using UML models and model-based simulations [20].

Scenario-based assessment methods have been described in the scientific literature and various experience reports validated these approaches. Among the most well known scenario-based we have the SAAM [15], the ATAM [16] and the SBAR (Scenario-based Software Architecture Re-engineering method) [1]. In addition, in a previous work we assessed one Nokia product family software architecture for evolution using a scenario-driven approach derived from the methods cited above [17].

Knowledge-based assessments are used when a model of the system that contains all the necessary information for the assessment is not available. The documentation of the system may exist but, the architecture is described in a format that does not provide enough information to extract all the interesting architectural properties. In these cases, the software may be assessed based on the knowledge that people have about the system.

In our case study, the main reasons behind our company's decision to evaluate the software architectonics using a knowledge-based approach were:

- the lack of formal and complete architecture documentation;

- the wide scope of the assessment;

- the number of the stakeholders involved;

- the geographical distribution of the development team.

The main contribution of this paper is the presentation of a case study and the lessons learned from the usage of a knowledge-based assessment method. The paper is structured as follows. First, we introduce the problem statement in section 2. In section 3 we present the case study. In section 4 we summarize the characteristics of knowledge-based assessments. The lessons learned are reported in section 5. The conclusion is in section 6.

2 Problem Statement

Technically, rigorous assessments should be based on a model. When a model is at hand (especially if based on an unambiguous modeling language) the assessment can be carried out by executing the model [20, 12]. In practice, however, a model of the software system to be assessed is not always available. This is often the case with software in the industrial realm, that runs a very complex system, undergoes frequent changes, has high variability and is based on legacy software. Often, the software architecture documentation that describes the system is not kept up-to-date for the very reason that the system changes too often and on a too tight schedule. Additionally, these software systems are maintained and developed in a distributed fashion and involve several organizational units that report to different lines of management. In many cases there is no centralized authority that deliberates over the evolution of software, and the system ends up evolving in a semi-controlled way (in the best of cases).

This situation generates a phenomenon that is sometimes referred to as architectural decay, or design erosion [24]. After the system has evolved for some time, a reliable model of the architecture cannot any more be obtained without a substantial reverse architecting effort [23].

In the case of very complex systems (order of several thousand architectural significant entities) undertaking a reverse engineering approach may not be convenient with respect to the benefit that the architectural assessment brings. All these conditions were verified, at least to a certain extent, in our case study.

Specifically, we focus on the problem of assessing change and evolution. During the life of a software system, the architecture evolves in order to support new requirements. When a potential new requirement is recognized, the owner of the software system must make a business decision whether or not to implement features that fulfill the requirement. Implementing them bears a cost and the tradeoff between the added value and the cost of implementation determines the outcome of the aforementioned business decision [22].

In practice, estimating the cost of implementation of a certain feature is not always straightforward. This is

one of the main reasons why evolutionary assessments are rather widely used by software development organizations. The research work that we expose concerns the assessment of the architectonics of a large software-intensive system [13], seen from the evolutionary point of view. The architectonics view that we aimed to achieve groups architecturally significant components (also referred to as entities) into layers that differ according to their ease of change (or cost thereof, which is almost always inversely proportional in a profit-making organization).

The specific case study concerned the software architecture of Nokia's Series 40 mobile terminals. Series 40 is a proprietary, in-house developed software platform that has been in use for over 10 years. It has been mainly deployed in low and middle-range products, as it combines ease of use with low memory consumption and lightweight processor requirements, both key factors in the cost of manufacturing a mobile terminal.

The Series 40 architecture was created by a handful of individuals in the mid-1990s, when mobile terminals where essentially portable telephones, with very few extra features. Although some of today's applications (such as messaging, browsing, phonebook) were already supported in the initial releases, several system-wide requirements simply were not conceivable at the time when the architecture was first implemented. Examples of such requirements are: color displays, enhanced phonebook, multimedia (camera, messaging, gallery, ring tones) and third-generation (high-bandwidth) protocols.

Most of these requirements could be incorporated into the architecture in a somewhat graceful way, and the impact of each one of them on the overall system architecture was relatively contained. However, after several years of successful expansion, we could not be certain that the architecture could withstand the advent of third-generation technologies and protocols.

Another problem was recognized to be the organizational structure of the development unit. For historical and business reasons, the development of the Series 40 architecture was spread into seven sites, located in three continents and separated by several time zones, language and cultural barriers and, worst of all, by a rather large number of management layers. The documentation process was not rigorously defined across all teams, and, although a centralized architecture management unit had been established, it was felt that some components were developing in an uncontrolled fashion. Overall, nobody could claim to have a full picture of the status of the architecture at any given time.

With new requirements imposed by the new standards, several of the paradigms that lied at the basis of the existing architecture were under threat. Third generation seemed to have a considerable impact on the very foundation of the architecture that we had successfully used for so many years. This fact, combined with the recognized lack of a formal and updated model of the architecture, warranted a business decision to undertake an extensive assessment of the whole software system.

The purpose of the assessment was to determine to what extent the Series 40 architecture would have to be changed in order to fulfill the basic requirements that the new standards brought. The result of this assessment would then be used in order to decide whether it would be economically more convenient to re-architect part or all of the system, as opposed to implementing a gradual evolution of the existing architecture. In a word: the evolvability of the architecture needed to be understood. We did this by producing a software architectonics view by means of a knowledge-based assessment.

3 Assessing the Nokia Series 40 software architecture

Our experiment focuses on the whole software product family architecture [14, 2, 5], including multimedia requirements, communication protocols and the support of new hardware for the whole range of products in the family.

Since an accurate model of the system architecture that represented the changeability (or ease of change) of the various system entities was not available when the assessment work was commissioned, we decided to undertake a knowledge-based assessment.

Knowledge-based software architecture assessments are based on the knowledge of the stakeholders involved in the software development organization. These stakeholders are the chief architects, architects, developers, requirements engineers, customers and, generally, all the people that are somehow involved in software development, testing or maintenance tasks.

Bosch [2] describes experience-based assessments as a way to evaluate and assess a software architecture. Our concept of knowledge-based assessment is derived and extended from Bosch's: we borrowed research methodologies from social sciences in order to make up for the absence of a rigorous model that could provide us with an accurate snapshot of the software system's architecture.

The method can be applied to any large software intensive system for the evaluation of the architecture and its evolution. Knowledge-based assessments can

be performed during the whole software life cycle, and their introduction in the software life cycle has advantages that will be described in this paper.

3.1 Assessment process description

Defining the scope of the assessment and listing the main stakeholders is the first phase in the process (called preparation phase). The second step consists of interviewing the relevant stakeholders (we call this execution phase). The analysis phase is the last one: the material that was collected during the assessment is integrated with information taken from the architecture documentation and subsequently analyzed and evaluated by the assessment team.

Iteration is implicit in the process since new people are added to the interviewees list during the execution phase, the architecture documentation is collected as the assessment proceeds, and the results are analyzed and refined continuously throughout the whole assessment.

3.1.1 Preparation phase: scope and list of the main stakeholders

In the initial phase, the purpose and scope of the assessment must be defined. At a broad level, this is usually done together with the customers of the assessment (i.e. the organizational unit or person that commissions the work). However, as more information is collected during the execution phase, the scope of the assessment is refined as more specific knowledge of the interesting aspects of software architecture and of the potential problems is gathered.

The definition of the purpose results in an executive summary for the assessment, usually a few paragraphs long, that is approved by the customers of the assessment. After approval, the assessment summary is sent to all relevant stakeholders in order to provide an overview of what the exercise is all about. In our case study, the purpose could be summarized as evaluating the impact on the Series 40 software architecture of a number of new requirements, all originating from the adoption of third-generation mobile telephony technology.

Having an executive summary at hand helps communicating the purpose of the assessment to a generally unaware audience. Not surprisingly, the knowledge about this sort of activity is scarce among the software development and testing community, at least in the organization where this experiment was carried out. The executive summary enabled us to sell the idea among time-stretched teams by communicating a common goal that everyone could recognize as useful.

Properly defining the scope of the assessment takes time. In our case, no individual (not even the people that were responsible for the Series 40 architecture as a whole) could provide a reliable and complete list of the requirements that needed to be considered in the assessment. For this reason, we decided to form a working group with the goal of defining the list of questions that would have formed the interviews during the execution phase. The group consisted of about a dozen experts in the various domains in the architecture: requirements, implementation, testing (or, in another dimension, protocol, user interface, hardware) and so on.

The work of this group consisted of two phases. First, we (the assessment team) presented the executive summary, summarizing the purpose of the assessment during a kickoff meeting. Then we assigned action points to each member of the group: their task was to come back with a list of a few (three to five) requirements that they thought would have an impact on the architecture, with a description of the reason why they thought each requirement was relevant to the assessment.

Subsequently, we elaborated the requirements and ranked them in order of importance based on the business priority and potential impact on the architecture. In doing this, we tried not to leave out any of the aforementioned domains, as this would have put the completeness of the assessment at stake. We finally turned all requirements into open questions (that is, questions that cannot be answered with a yes or no statement). A typical question could read, say: "how do you think the architecture of the protocol state machine that we have should change as a consequence of the adoption of the WCDMA protocol family?". Before finalizing the list of questions, we submitted it to the owners of the assessment and the expert group for review, which took place during another meeting. At the end of this meeting we had an approved scope.

The process of scope definition lasted approximately one working month, and produced a document that listed around twenty major requirements that were thought to impact the architecture.

After defining the purpose and scope of the assessment, the next step consists of defining the list of the interviewees. The chief architect, as well as the experts that formed the initial scope definition working group, all had a high-level view of the software development process and of the responsibilities in the organization. This is why we turned to them again in order to define the initial list of persons that were to be interviewed.

This list was fundamental to start but it was not definitive. During the interviews, new people were

added to the list whenever the need to investigate further topics emerged.

We have divided the stakeholders in three broad categories to be interviewed.

- Requirement engineers, who are responsible for collecting forthcoming requirements and, in some cases, evaluating their potential impact on the architecture.

- Architects, who are responsible for the design of some part of the software architecture.

- Developers, testers and experts in some key quality attribute, such as performance. These are the more "hands-on" people.

The categories must be structured in order to cover all stakeholders within the scope of the assessment.

During the first phase, 18 persons were listed, together with their title and responsibility within the organization. Those were the persons to be interviewed first, and mostly belonged to the first category (requirements engineers). However, during the whole process the list of interviewees steadily grew as we were pointed to other experts, and the final list of stakeholders numbered 35.

This method enables repeatability in different organizations. In case a different organizational structure is in place, the assessment team must evaluate it and come up with a good way of covering the majority of relevant people in the organization. A good principle to be kept in mind is that the assessment team must include, or be supported by, someone who is very familiar with the organizational structure of the software development units. This is more important as the size of the organization (and that of the software system) grows.

In practice, the assessment was carried out by interviewing all people in the first category before continuing with the second category, and so on. However, in cases when new people were added to the list, or when the development team was distributed geographically, for practical reasons we made an attempt to interview as many people that were based in the same site as could be allowed without compromising the results of the assessment.

3.1.2 Execution phase: interviews

Due to the high number of interviewees, and to the fact that usually they were involved in projects with tight timelines, we made a decision that the assessment team members must travel to the sites where the interviewees were based. This would minimize costs, as well as gain in efficiency from the software development point of view.

The assessment scope and methodology was presented to the interviewees at the beginning of each interview. All interviewees were asked for comments on the assessment scope and methodology. We used those comments to refine the executive summary and improve the assessment method, even though we made a deliberate choice to stick to the agreed general structure in order not to compromise the validity of the study (or have to repeat certain interviews due to changes in the method).

The interviews were semi-structured, in the sense that we gave the interviewees a reasonable margin to discuss the main concerns in the architecture from their point of view, independently of the questions. However, we made a point in guiding the interviewees and trying not to lose the focus of the assessment. This task is always problematic when facing world-class experts and following semi-structured interviews. This is why the presence of at least two members of the assessment team was of vital importance for the success of the experiment.

During the interviews, the assessment team took notes, which in turn went to feed into the final report. In this phase, it is essential that anonymity is guaranteed: it must be clearly stated that the final report will not contain any reference to the name of the persons being interviewed, only technical finding will be reported to the management. The statement ensures that the interviewees are free to report their opinions on politically sensitive topics without any fear of compromising their career.

In our case study we carried out interviews with 35 people who were based in nine sites that were located in five countries. Each interview lasted from 40 to 80 minutes, depending on the breadth of the competence that the interviewee had (and thus, on the number of questions that made sense to be asked). The execution phase lasted approximately three working months. All interviews were carried out in English.

3.1.3 Analysis phase

In our method, analysis is done gradually as the interviews proceed rather than in one shot at the end of the work. The material collected is reviewed after each interview and the analysis is revised and refined during the whole process. New information and documentation is collected continuously during the assessment. In case no information on a certain part of the architecture is available, the assessment outcome must highlight this fact.

The interviews represent the point of view of the stakeholders. However, they must contain factual evidence of the claims and must be grounded from the technical point of view. Often, opinions can be biased toward the architecture (for instance, interviewees were allowed to criticize a particular design choice). When this occurs, the work of the assessment team during the interviews is of guidance and inquiry for references and reasoning for criticism.

During the analysis work, only technical findings are evaluated and unsubstantiated opinions are discarded. In some cases, the responses can highlight the need for a more accurate analysis. After all, software development is a human-driven activity where experience is an important asset and not all technical choices can be motivated with hard facts.

The analysis results into a technical report that highlights all open issues and suggests directions for research and architectural improvements, which may be undertaken as follow-up activities.

We structured the report by following the same categorization as the initial list of questions. For each issue we presented a problem statement, a description of the issue and, when it was deemed that the issue did not need further studies, we highlighted the proposed solution. In addition, the report contained a wishlist which emerged from the interviews. The wishlist, the list of desired features raised by the stakeholders, was also an important outcome of the assessment.

In general, the final report suggested improvements (rather than just pointing out issues as described by the interviewees) only when unanimous views on how to solve a certain issue had been expressed by several people who did not all work for the same team.

Subsequently, the report was delivered for comments to all the interviewees, as well as to the expert group that helped scope the study. Comments and corrections were encouraged in this phase. This helps correct typos and misunderstanding that almost inevitably occur when performing semi-structured interviews in a foreign language (most of the participants to the experiment did not speak English as a first language, and nor did the members of the assessment team).

After the review, the final report was delivered to the customer of the assessment, who organized a wrap-up meeting with the initial expert group. During this meeting, the open items were listed and commented in a brainstorming fashion, though no changes to the report were made. After discussion, action points were given to relevant people in the organization.

The final report contained 65 action points and the open issues were prioritized and ranked. For every action point a responsible person and a closure deadline was assigned. The owners of the action points were then free to handle the work together with the management as well as with the assessment team, who could point to interviewees in case clarifications were needed. We still tried to preserve privacy, though at this point the benefits of the assessment were generally understood and people were generally happy to be involved.

After the final assessment report was approved, we obtained a view of the architectonics of the Series 40 software by ranking the existing architectural entities by their capability to support the requirements that constituted the scope of the assessment. The view was formalized in form of a list of the various entities. Every entity was assigned a judgment of its capability to evolve in sight of those requirements. We limited the number of categories to three (high, medium and low capability to support the requirements), as obtaining a finer grain ranking was difficult due to the qualitative nature of the study.

The architectonics view was then delivered to the customer as part of the final report. This helped the architecture management team to identify the critical areas of evolution. Funds for further analysis work (corresponding to the action points that were mentioned in the previous section) were then allocated in a way that reflected the architectonic categorization. The view was simple and by nature error-prone, as drawing a limit between the three categories was largely subjective. However, we found out that the architectonics view represented a useful communication tool towards upper management, since its simplicity allowed it to be understood by non-technical people who were not familiar with the details of the architecture and had no chance to go through and understand the seventy-odd-page report.

4 Characteristics of knowledge-based assessment

Knowledge-based assessment is centered around people and what they know. The strengths of this assessment method are the experience and the knowledge of the stakeholders involved in the creation of the software. Software creation is still fundamentally a human activity and the people involved in the development process are the key persons with the experience and the knowledge to improve the software architecture.

The input for the assessment is the material collected from the interviewees by the assessment team and the available documentation about the architecture. Given the limited time and the potential large number of people in the software development organization, only a limited number of stakeholders can be

interviewed. In our case study, the target organization was made of over one thousand people distributed in various sites around the world. It follows that focus and the selection of the stakeholders is fundamental.

The assessment method is particularly useful to evaluate the evolutionary path of the software architecture during the whole life-cycle. Since it does not require a complete architecture documentation in any particular form, it can be applied in the early stages, when the software architecture is not completely defined. Additionally, legacy systems with no formal architecture description can benefit from these assessments. The fact that a well defined formal software architecture description is not used differentiates this assessment method with model based assessment methods. Model driven methodologies have the ability to automate and explore the effects of change decisions by using model transformations and examining the effects on the software [20, 12]. However, a model of the system is not always available, especially in large, rapidly evolving software systems.

The ability to evaluate and mitigate the risk of software evolution in time, before the software is developed, is fundamental. Therefore, assessments must be included in the software development life-cycle and used especially when major changes due to forthcoming requirements are planned.

The resources required for this kind of assessment depend on the size of the software development organization and on the scope of the assessment. Large software organizations where development teams are distributed geographically are the natural setting for knowledge-based assessments. The large number of stakeholders differentiates from scenario-based assessments [17, 16, 15, 1, 4], where brainstorming meetings are used. Scenario-based assessments have proved to be very effective, however, the stakeholders should not exceed the reasonable size for a meeting in a room.

The outcome of the assessment includes the current overview of architecture, the main issues found, and optionally recommendations for their resolution. Often, follow-up activities and additional assessments are started as consequence of the final report. This document also indicates possible areas where additional investigation is needed. Quantitative assessments can also be used to model and simulate software evolution. For example, software performance engineering (SPE) [26] can be used to estimate and model the software performance of a software system. We have used the SPE methods and quantitative assessments to evaluate the same software platform reported in this case study [6, 8, 10, 9, 7].

5 Lessons learned

We used the experience that we gathered in performing the case study in order to learn and improve the very assessment process. The assessment was recognized to be useful inside our organization and these assessments are now performed regularly. In the subsections below we list the lessons learned.

5.1 The Scope Must be Well Focused

We applied knowledge-based assessment to a large software system that is developed in a geographically distributed manner. Assessing the architecture for evolution was our target. Having such a wide scope puts the ability to deliver a concrete and useful outcome at risk. Therefore, it is important that the scope is well defined before the assessment starts and that it is agreed with and supported by the person commissioning the work. During the assessment, the scope can be refined, but the objectives and scope must be consistent throughout. Introduce the assessment scope and the methodology before each interview was useful in this perspective. This permitted a clear understanding of the scope and outcome of the assessment to the interviewees. In addition, it highlighted the real impact and importance of their contribution.

5.2 Documentation Improvement

The main outcome of assessment is a document that gives an overview of the current architecture and the main areas where new requirements are likely to impact. Additionally, during the assessment it is possible to highlight areas where the architecture documentation must be improved, or completely lacks. In a continuous evolving systems, the architecture documentantion does not perfectly reflect the actual implementation. Furthermore, new architectural components may be in the process of being defined and consequently, a well documented and specified architecture is not always achievable. Therefore, one of the indirect benefits of assessment is the identification of areas where documentation needs attention.

5.3 Communication Improvement

Communication is one of the main challenges in large and complex organizations such as the one where this experiment was carried out. The assessment report document represents a shared view of the architecture and an analysis of its evolutionary path.

In our case, nobody in the organization had accurate and complete knowledge of the software architecture. The document provided a snapshot of the various software components and architectural dependencies. It was revised by the stakeholders involved and inaccuracies were corrected. The final version was also distributed internally and provided a high level picture of the system. As such, it made for an excellent communication tool, in that it conveyed a commonly agreed view of the architectonics of the system, and its potential deficiencies.

The improved communication was also the consequence of interviewing people in different parts of the organization, working in different sites and in different areas of expertise. The assessment team had the opportunity to report issues raised by requirements managers, and then interview the technical experts on them. Misunderstandings are a consequence of communications problems, cultural differences and the distribution of the development teams. As example of communication issues, in one case, some stakeholders raised a potential problem that had actually been fixed months earlier. Consequently, one of the valuable assessment outcomes was the improvement of the communication.

5.4 Use only Technical Findings and Reasoning

As matter of fact, in knowledge-based assessments, people report their own view on the software architecture problems and evolutionary path. People tend to be optimistic and pessimistic. Optimistic people will have the positive attitude that everything can be solved and no major requirements impede the software evolution. On the other hand, in pessimistic persons a negative view on the software architecture evolutionary path may be predominant.

When personalities come into play, it is always challenging to split biased opinions from hard facts. The assessment team must limit the discussion to technical details and stick to facts rather than let the opinions of the interviewees prevail. The interviewees assertions must be technically grounded and the assessment teams must guide the interviews accordingly.

For example, it is not enough to assert "Linux will solve all our problems with user interface". If such an assertion is not motivated, interviewers must seek for facts that corroborate it. They can do so by asking questions such as "what improvements would Linux bring and for what reasons¿'

The assessment team then reviews all notes taken during interviews, double checks on dubious facts and analyzes the provided statements. This said, intuition and experience of the stakeholders are valuable assets for a company, and additional quantitative assessments or analyses may be required before opinions are discarded as irrelevant.

5.5 Assessment team not part of the development team

In our case study, the assessment team was composed of three researchers from the Nokia Research Center, a separate corporate entity from the Nokia Mobile Phones business unit. The assessment team was responsible for scoping the assessment, organizing, scheduling and conducting the interviews, refining and ranking the assessment topics, as well as publishing the main findings.

The assessment team had a certain degree of experience in the architecture and an understanding of the general principles that the architecture was based on. However, they were not as knowledgeable as the people involved in the actual development unit.

Having an assessment team that is not part of the development team has advantages and disadvantages. On one hand, an external assessment team does not have a deep knowledge of the system to be assessed, and often cannot evaluate the correctness of statements that are made by interviewees. On the other hand, an external team do not have vested interests in promoting a biased evaluation, and do not have assumption over past design decisions.

Furthermore, having an external team and preserving the anonymity of the interviewees encourages stakeholders to speak freely. For example, a developer may fear the superiors or may not like to criticize the work of a colleague in public. However, at the end, only technical details are analyzed and included in the report. This increases the completeness and fairness of the final report.

6 Conclusion

Assessing the current software architectonics and its evolution is the main objective of the knowledge-based assessment method that we present. The method gains its strengths from the experience of the stakeholders involved in the process. The main input of the process is the material that is collected by interviewing selected stakeholders, as well as from architecture documentation. Large, software-intensive systems are a typical application of knowledge-based assessment methods.

We have used a knowledge-based approach to evaluate the architecture for evolution in a complex environment. Large software systems with software develop-

ment teams distributed in different time zones and with cultural differences represent several challenges. However, the assessment proved to be effective and valuable within our organization.

Further research is needed on the comparison between an interview-based methods (such as the one we used) and brainstorming-based methods (such as SAAM, ATAM and SBAR [16, 15, 1]), in situations where it is feasible and convenient to bring most stakeholders to a unique face-to-face meeting. Finally, we warrant further studies that prove the applicability of the method to focused assessments, where the subject to be analyzed is more narrow than the one we had to face.

References

[1] P. Bengtsson and J. Bosch. Scenario-based software architecture reengineering. *Proceedings of the Fifth International Conference on Software Reuse (ICSR)*, pages 308–317, June 1998.

[2] J. Bosch. *Design and Use of Sofware Architectures.* Addison Wesley, 2000.

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

[4] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architecture.* Addison-Wesley, 2002.

[5] P. Clements and L. Northrop. *Software Product Lines.* Addison Wesley, 2002.

[6] C. Del Rosso. The process of and the lessons learned from performance tuning of a product family software architecture for mobile phones. *Proceedings of the 8th European Conference on Software Maintenance and Reengineering*, Tampere, Finland, March 24-26 2004.

[7] C. Del Rosso. Dynamic memory management for software product family architectures in embedded real-time systems. *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 211–212, 2005.

[8] C. Del Rosso. Performance analysis framework for large software intensive systems with a message passing paradigm. *Proceedings of 20th Annual ACM Symposium on Applied Computing, track Embedded Systems: Applications, Solutions, and Techniques (EMBS), Santa Fe, New Mexico, March 13 -17, 2005*, November 2005.

[9] C. Del Rosso. Experiences of performance tuning software product family architectures using a scenario-driven approach. *Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering (EASE), British Computer Society*, pages 30–39, April 2006.

[10] C. Del Rosso. Reducing internal fragmentation in segregated free lists using genetic algorithms. *Proceedings of the 2nd International ACM Workshop on Interdisciplinary Software Engineering Research*, pages 143 – 150, 2006.

[11] R. L. Glass. *Facts and Fallacies of Software Engineering.* Addison Wesley, 2003.

[12] J. Gray, Y. Lin, and J. Zhang. Automating change evolution in model-driven engineering. *IEEE Computer*, 39(2):51–58, February 2006.

[13] IEEE 2000. Recommended practice for architectural description of software-intensive systems. *IEEE Standard No. 1471-2000*, September 2000.

[14] M. Jazayeri, F. Van Der Linden, and A. Ran. *Software Architecture for Product Families.* Addison Wesley, 2000.

[15] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, pages 47–55, November 1996.

[16] R. Kazman, M. Klein, and P. Clements. Atam: A method for architecture evaluation. *Technical Report CMU/SEI-2000-TR-004*, 2000.

[17] A. Maccari. Experiences in assessing product family software architecture for evolution. *Proceedings of the 24th International Conference on Software Engineering, (ICSE) 2002*, pages 585 – 592, 2002.

[18] A. Maccari and G. H. Galal. Introducing the software architectonic viewpoint. In J. Bosch, W. M. Gentleman, C. Hofmeister, and J. Kuusela, editors, *WICSA*, volume 224 of *IFIP Conference Proceedings*, pages 175–189. Kluwer, 2002.

[19] N. Medvidovic and N. Taylor, Richard. A classification and comparison framework for software architecture description language. *IEEE Transaction on Software Engineering*, 26(1):70 – 93, January 2000.

[20] Object Management Group (OMG). Model driven architecture (mda). http://www.omg.org/mda/, 2005.

[21] T. M. Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment.* John Wiley & Sons, 1996.

[22] A. Ran and R. Lencevicius. Making sense of runtime architecture for mobile phone software. *Proceedings of the 9th European software engineering conference, (SEC)*, 28(5):367 – 370, September 2003.

[23] C. Riva. View-based software architecture reconstruction. *PhD disseration, Vienna University of Technology*, October 2004.

[24] C. Riva and C. Del Rosso. Experiences with software product family evolution. *Proceedings of the sixth International workshop on principles of software evolution*, (SE-5 2), 2003.

[25] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual, second edition.* Addison-Wesley, 2005.

[26] C. U. Smith and L. G. Williams. *Performance Solutions.* Addison Wesley, 1995.