Software Architecture at Work
Technical Report # 5
aSQA: Architectural Software Quality Assurance

Henrik Bærbak Christensen
Klaus Marius Hansen
Bo Lindstrøm

June 2010

Abstract

In this paper, we present a novel technique for assessing and prioritizing architectural quality in large-scale software development projects. The technique can, after an initial investment, be applied with relatively little effort by software architects and the technique is therefore suited for agile development in which quality attributes are assessed and prioritized within each development sprint. We report on the benefits and liabilities of the technique based upon interviews and experiences from three companies and supplemented by a survey study. In conclusion, the technique is considered valuable and a viable tool, and have benefits both at the architectural, technical, level, as well as in a business and people context.

1 Introduction

Software architecture is a major concern in any large scale software development and therefore a central aspect for successful IT companies. One particular task that rests heavily on a software architect's shoulder is *architectural quality assessment, prioritization, and conformance checking*. It is well established that different architectural qualities often compete, the classic trade-off between performance and maintainability being one example, and that implementations of architecture may drift away from the architect's stated design due to misunderstandings, communication problems, and developers' skill sets. Software architects are thus in need of models, techniques, and tools to help in facing these challenges. Researchers and practitioners have responded by developing different architectural quality frameworks and evaluation techniques, a short outline is presented in the related work section of this paper. However, most of these techniques are characterized by being rather costly to perform (e.g. scenario-based techniques such as the Architecture Tradeoff Analysis Method (ATAM; [14]) and/or resulting in quality measurements that are difficult to

compare and thus form a poor basis for prioritizing effort (e.g. specific measurements as defined by ISO/IEC 9126 [11]). Thus they are less than optimal for software architects to apply continuously in agile development as well as to allow prioritizing which components and which qualities to focus on during a sprint.

The *architectural Software Quality Assurance* technique (aSQA), has been developed by software architects in Systematic A/S as a light-weight technique for continuos quality assessment and prioritizing in software architecture and development work. Systematic A/S [18] is a privately owned software development company in Denmark, employing approximately 500 people, 50 of these in the UK, Finland, and USA. Systematic is certified with respect to process maturity at CMMI level 5 and combines CMMI with lean development [16, 12], specifically the Scrum method [17].

The main contributions of this paper are twofold. First and foremost this paper presents the aSQA as a novel and viable technique for continuously assessing, controlling, and balancing quality attributes in a development project. Secondly, we provide case studies from three companies and qualitative evaluations of the method, that generally support the claims of the method as a light-weight and effective way of assessing and prioritizing software quality.

The paper is organized as follows. Section 2 outlines alternative software architecture evaluation techniques. In section 3 we describe the aSQA technique, while section 4, 5, and 6 present case studies of using aSQA in Systematic, two other companies, and finally a survey study. We discuss our findings in section 7 before we conclude in section 8.

2 Related Work

Much work has been done on techniques for evaluation of software architecture [10, 9]. Dobrica and Niemelä [10] surveyed eight software architecture analysis methods all focusing on predicting the effects on software quality on choosing a specific software architecture design. They categorize evaluation methods as (following [1]) either being "questioning techniques", based on qualitative, often software quality-independent questions to be asked of an architecture, or "measuring techniques", based on quantitative, often software quality-dependent measurements to be made on an architecture. In this spectrum, aSQA is somewhat agnostic in that it focuses on software qualities, but allows for both questioning and measuring techniques in analyzing a software architecture/system under development with respect to a given quality. In practical use, aSQA has typically been used with qualitative techniques.

Arguably, the most prominent techniques for software architecture evaluation are scenario-based following or based on SAAM [13] and ATAM. Babar and Gorton [3] compare four such methods and present a framework for comparing evaluation methods. The framework contains the components of context (e.g., goal of the method, quality attributes covered, development phase), stakeholders (e.g., stakeholders involved, process support, resources required), contents

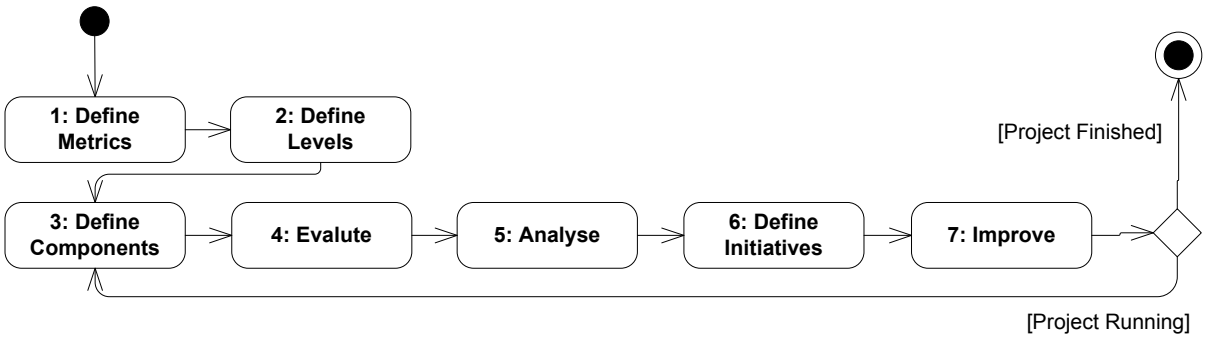

Figure 1: aSQA Process Steps

(e.g., activities and approaches), and reliability (maturity and validation). The aSQA technique has as a goal to make decision makers aware of the quality status of a software system throughout development, focusing on the quality attributes of a chosen quality framework. In particular, the focus on continuous assessment distinguishes it from other architecture evaluation methods. The technique is supported by a defined set of activities, a database-backed tool, and requires few resources to perform (see Section 4). Finally, the technique has been used for several years at Systematic and has matured in that way and has been evaluated through case studies at other companies (see Section 5) and in a survey study (see Section 6).

Recently, focus has been put on the ability for architecture evaluation methods to scale [19, 4] particularly in the context of Ultra-Large-Scale systems [15]. Zhu et al. [19] describe how to scale methods by combining process components from existing methods and Babar et al. investigate distributed evaluation (through distributed involvement of stakeholders). The aSQA technique is, first, lightweight and can, secondly, be applied in a hierarchical manner, both of which contribute to scalability. Furthermore, the technique has been validated (and developed) in the context of large systems in the healthcare and military domain.

3 The aSQA technique

The aSQA was developed to serve three different goals.

- *Allow continuous quality assessment.* Agile processes put emphasis on speed and functional requirements. An architectural assessment technique that could keep up with the high pace was needed. This necessitated a light-weight assessment technique that required only a small investment in time for the software architect and/or the development team.

- *Allow quality to be quantified.* In projects much focus is on easy quantifiable and measurable parameters such as cost, schedule and functionality.

However, many architectural aspects of quality are much more difficult to measure, like availability or maintainability. This makes it difficult to manage quality not only with respect to defining and communicating a goal for the quality of the product but also with respect to controlling if the intended quality goal is obtained. This necessitated defining a metric for quantifying quality attributes as well as championing a set of quality attributes as the set to consider.

- *Allow quality to be prioritized.* During development the software architect has to prioritize certain quality attributes in certain components before others to ensure costly development time is invested wisely. However, the metrics used for quality attributes vary greatly and makes comparisons highly difficult: is achieving 5000 transactions per second better than lowering estimated time to introduce a new taxation policy to 16 staff hours? This necessitated defining a uniform scale across quality attributes.

The result of these challenges is the architectural Software Quality Assurance (aSQA) that targets continuous evaluation of a software architecture for a system under construction with a focus on fulfillment of quality attribute requirements. The method can be classified as a metrics-based measuring technique according to the framework of Clements et al. [8], but it is also possible to use quality attribute scenarios [6] as a basis for evaluation, though this has not been pursued by Systematic.

The activities of aSQA are shown in Figure 1. The first step of aSQA is to define metrics to apply to components of the software architecture, requiring the choice of a quality framework (Step 1). The ISO/IEC 9126 standard and Bass et al. [6] are two examples of such frameworks. ISO 9126 defines metrics for internal quality, external quality, and quality in use that may be used (or built upon) in aSQA whereas Bass et al. defines quality attribute scenarios that may define the quality needed for components.

A crucial step in aSQA is to define how measurements made using the defined metrics map to levels (Step 2). In aSQA an ordinal scale of values ranging from 1 to 5 (a reference interval scale) is used ubiquitously since it allows a coarse and manageable comparison of values across quality attributes. An example of how to define levels is:

1 *Unacceptable:* Important stakeholders find the system unacceptable because of quality level of the attribute in question.

3 *Acceptable:* No relevant stakeholder finds the system unacceptable because of quality level of the attribute in question.

5 *Excellent:* All relevant stakeholders are highly satisfied by the quality level of the attribute in question

A prerequisite for using the evaluation part of aSQA is the existence of (a design of) components for a software system (Step 3). This definition is usually made in an iterative and possible incremental process in which the set of

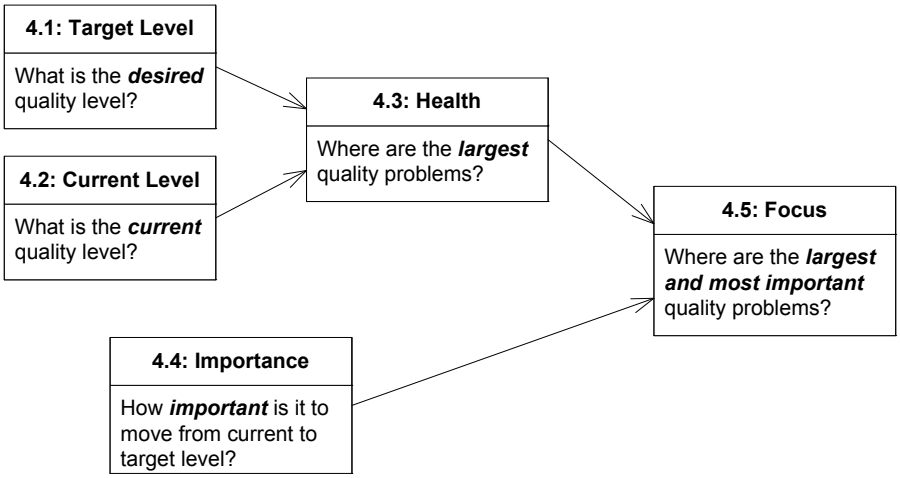

Figure 2: aSQA Evaluation Substep

components may change. The component structure and scope should preferably be stable and it should be meaningful to assign a level of quality to that component.

The actual evaluation is carried out in several substeps (cf. Figure 2). First (Step 4.1), a *target* is set for each component. The target is set according to what is currently seen as the needed quality level for a specific component. Secondly (Step 4.2), the *current* quality level is measured (using the criteria set up in Step 2). Together this gives the current *health* for components in the project (Step 4.3). Figure 3 shows a color-coded example of calculating levels for an imaginary architecture containing three components, Terminal, Scanner, and Application Server, and using the six system quality attributes defined by Bass et al. [6]. The specific calculation performed for health follows the following formula (that assigns health according to how far below the target level the current level is):

$$health = 5 - max(0, (target - current)) \tag{1}$$

The health values are then compared with the defined importance levels (Step 4.4) that are assigned according to which quality attributes and components are prioritized. Combined, health and importance determine the focus levels of the project (Step 4.5). In the figure, the focus level is calculated as:

$$focus = ceil((6 - health) * importance/5) \tag{2}$$

The lower health is and the more important the quality attribute is for the component, the more focus there should be on raising the level of the quality attribute for the component. In the example, particular focus should be put on performance for the Application Server (since its focus level is 5). The final steps of an iteration of aSQA is to analyze the results (Step 5), based on that

	Terminal					Scanner					App Server				
	t	*c*	*h*	*i*	*f*	*t*	*c*	*h*	*i*	*f*	*t*	*c*	*h*	*i*	*f*
Availability	4	4	5	1	1	3	3	5	1	1	4	4	5	1	1
Performance	5	2	2	5	4	5	3	3	5	3	5	1	1	5	5
Modifiability	4	4	5	2	1	3	4	5	2	1	5	5	5	2	1
Testability	4	4	5	2	1	3	4	5	2	1	5	4	4	2	1
Security	4	4	5	2	1	3	3	5	2	1	5	2	2	3	3
Usability	5	1	1	2	2	5	2	2	2	2	3	3	5	2	1

Figure 3: aSQA Levels Example. Shown are target (t), current (c), health (h), importance (i), and focus (f) levels.

define initiatives to improve quality (Step 6), and finally to do the improvement (Step 7). The process step 4 is supported by a tool that presents overview in tabular format as exemplified by figure 3.

4 Case studies at Systematic

Systematic has adopted ISO 9126 as their quality framework. When performing aSQA, Systematic does not distinguish between internal and external quality but also consider quality-in-use. This means that "functionality", "reliability", "usability", "efficiency", "maintainability", "portability", and "quality in use" characterize the overall quality of Systematic's products. Even if ISO 9126 has been criticized of not being complete and ambiguous (e.g., [2], in Systematic's experience, the standard and selection of characteristics have tended to cover relevant quality aspect even if coarse-grained. Each of these seven characteristics are further divided into sub characteristics according to definitions in ISO 9126; and in particular efficiency (and time behavior as a sub characteristic) is important. Most projects now use aSQA, which was introduced in 2005, and perform an evaluation on either a monthly or quarterly basis depending on the type of project.

At Systematic, we had two interview sessions with architects, focussing on two cases of use of aSQA:

1. Use in a product setting. Systematic develops products that are either sold directly as COTS products or the products are customized by Systematic to the needs of a specific customer. We interviewed the architect of these products

2. Use in a contract development setting. A main part of Systematic's work is contract development of which their Clinical Information System (CIS) is an example. The system is a large enterprise system used across several hospitals in Denmark. It has been in operation for six years and development of extra features and modules are still going on. We interviewed the lead architect of the CIS system

The interviews were open-ended focusing on how aSQA was used, the effort in using it, and the benefits and liabilities in using the method. Furthermore, we draw on previous ethnographic studies among others at Systematic [7]. In the following, we report from these interviews.

4.1 aSQA in Product Development

Systematic develops a number of different defence related products in a group with 50-70 developers. The architect we interviewed was architect on two strategic products, SitaWare and IRM, ("SitaWare" is a military command and control system while "IRM" is a Multilateral Interoperability Programme (MIP)[1] replication mechanism with the purpose of establishing interoperability between command and control systems) and consultant on the remaining products. He performed aSQA for all products.

At the time of the interview, aSQA was performed every three months, the result being part of a technical status report used by management and architect for planning of sprints. Previously, aSQA was performed monthly, but since few changes are made on an architectural level currently the frequency was reduced. In this case, since the products were still under development, this might be an indication that the selected components are too coarse grained.

For a report, the aSQA process takes 4-6 hours for the architect. For the products that the architect is not involved in on a daily basis, the lead developers take part in the aSQA evaluation which takes less than one hour.

The goal levels for the individual products are set by project management and are often constant over a long period of time but may change as visions for products evolve. Often the goal level (e.g., 4) does not change, but the requirements behind the level may change; it may, e.g., be required that a product can triple the number of supported clients. In this case, each product is treated as one component in the evaluation because of the number of products.

How to measure the current levels of quality is up to the architect. In the product case, analyzability is e.g. measured through a combination of (running) conversations with developers and time behavior is measured through test scenario runs. For each of the products, different characteristics may be important. For SitaWare, e.g., "interoperability" (a sub-characteristic of "functionality") is highly important (since it is used in a setting with different military coalition partners) and this sub-characteristic is measured in integration test events.

With respect to focus, importance is often based on a product road map and combined with current level this provides input to project management. Project management does not use the health levels from aSQA directly, but rather a summary made by the architect. Based on this, the top three foci are often addressed in the next sprint, but prioritization is also made in other ways by management.

The effect of using aSQA has been for the architect to be able to have an overview of a complex set of products. Primarily, this effect comes from

[1] `http://www.mip-site.org`

being attentive to quality continuously, but having deadlines (for producing a report) is also important. Furthermore, the quality framework has given a way of communicating quality with developers. Here, it is primarily the characteristics of quality (and not the sub-characteristics that are used).

One issue in using aSQA for platform products (such as SitaWare which is also used in contract development at Systematic) is that ratings are tied to stakeholders. This means that projects that use Sitaware may have usage scenarios different from those that are envisioned by the Sitaware architect. If this is the case the experiences from such projects are often used as input to future aSQAs of Sitaware – if the quality goal of the product has been decided to be raised or if the product roadmap which defines the overall strategy and target of the product has changed. One example is a project that uses low-bandwidth radio communication for messaging. In this case, tests of time behavior of the SitaWare product has been amended with simulations of radio communication.

4.2 aSQA in Contract Development

The CIS of Systematic is a large project, involving 70 developers developing ten different "modules": seven "user modules" (e.g., for medication or requisition) and three "platform modules" (client-side, server-side, and report generation). We interviewed the lead architect for healthcare who leads a group of four architects and three domain experts with healthcare expertise.

The project develops a single integrated CIS system that is used in multiple hospitals. Some features (such as integration with a medication robot and PDAs) may only be used in one location, but are part of all installations.

The aSQA is used as a coordination tool across the modules of the project. Each module has an assigned architect and domain expert that perform aSQA on that module together. Following this the full group (seven persons) meet and produce a full aSQA for the complete CIS system. The result is a health and focus level for each quality attribute (for each of the ten modules) which is used in a dialogue with project management for deciding on priorities of addressing potential quality issues.

Every three months, an aSQA is performed in which importance is set in accordance with what is to be delivered next in the project. At the time of interview, the project had just performed their fourth aSQA in which about 40 hours is used on the evaluation. According to the lead architect this is a reasonable level of effort to be used.

Quantitative measurements only form a small part in setting the current level of a characteristic whereas the judgement of the architects and domain experts form the rest. An example is usability in which pilot tests, user experience groups, and participatory design with end users all give input, but where the judgement of current level of usability is made (primarily) by the domain experts.

The lead architect also in this case points to that one of the big advantages of aSQA is the need to choose a common quality framework that can be used for communication between architects. Furthermore, it is possible to be very

concrete on what a quality goal is and where a quality problem is in terms of (sub-)characteristics. In the contract for development, goal quality has been primarily defined as "good enough for initial deployment and use" and the aSQA process has contributed highly to a more precise definition.

One issue is that it is unclear whether aSQA scales to the sub-projects that develop the individual modules because of the effort needed on these. aSQA on the other hand provides a frame for the quality requirements of the features developed in the sub-projects.

5 Case studies outside Systematic

Systematic A/S was part of the research project, *Software Architecture at Work* (SA@Work) [7]. The SA@Work project was a a one and a half year project that started August 2007 and ended December 2008. It was a research collaboration between two of the authors and four Danish IT companies. Two other companies agreed to test the aSQA technique within their own context. The two companies were:

1. *Bang & Olufsen (BeO)* produces high end audio products, television sets, and telephones. The company was founded in 1925, in Struer, Denmark. Since the beginning the company has focused on creating quality products. The software development organization of B&O consists of offices in Struer and Aarhus (Denmark) and a subsidiary in Estonia, furthermore they work with a consultancy company in India.

2. *Jyske Bank (JB)* is the second largest independent Danish bank, employing some 4,000 people in 119 Danish branches. The software development organization of Jyske Bank is the largest in Jutland, designing, implementing, and running the systems of Jyske Bank in addition to processing central-government payments and operating a payroll system for the Danish counties.

The testing was agreed on a meeting May 2008 in which the aSQA method was presented by a software architect from Systematic and a strategy was discussed. It was agreed that the two test companies would get the associated software installed and architects would find small pilot projects to test the aSQA technique on.

Due to resource constraints within the research project, a small scale qualitative study was adopted. The researchers would act as method consultants for the two test companies and the testing would be concluded by an interview. The researchers would prepare a questioning guide but otherwise the interview of the architects would be open ended to allow impressions and novel contributions to be identified. Thus the application of aSQA would be small-scale in the test companies and the tracking of the process by the researchers relatively lightweight. Nevertheless some interesting observations and alternative uses of the model showed up, as outlined below.

5.1 The Interview Guide

The interview guide was structured along three dimensions: hypothesis, concerns, and concrete context.

- The hypothesis contained three central postulates that we wanted the test companies to assess.
  - *aSQA forces an explicit focus on architectural qualities.*
  - *Architectural assessments are made continuously (for instance at the end of each sprint)*
  - *aSQA is a light-weight technique (small investment in time and resources)*
- The concerns were about granularity of components, component stability, and whether the results of aSQA matched architect's "gut feeling."
- The context part contained detailed questions concerning each of the steps in the process (figure 1), both on a technical, practical, level as well as evaluation of which aspects posed hard challenges or were ambiguous.

5.2 aSQA at Bang and Olufsen

An architect from Bang and Olufsen was interviewed late October 2008. The primary focus of the particular team was a developing a framework for audio-video processing. The outcome of the component selection step was that they chose to evaluate the framework as a whole, i.e. a course grained selection leading to a single component. Regarding quality model and scale, they simply adopted those proposed by Systematic. The architect and his team evaluated the current level while the target and importance levels were determined in a workshop with the *product owners* i.e. stakeholders that are supposed to build complete products using the framework. Regarding current level, several architects and developers did the assessment independently and came to rather different results. A single meeting was initiated to classify the reasons for the discrepancies and come to a single and generally accepted result but ended in three meetings before a common understanding was reached. People interpreted both ISO qualities as well as the scale rather differently, but the discussions were fruitful in creating a common understanding.

With regards to the hypothesis, the impressions of the architect were positive, and basically supported the postulates set forth in the interview guide. A major problem for the team developing the component evaluated has been that development is part of a very long term strategic effort, and the product owners' perception of what is important has a tendency to change over time. This leads to many functional and architectural requirements being unspoken and thus subject to renegotiations at every meeting. The aSQA's focus on concrete numbers and formalized goal levels *"forced the stakeholders to make up their minds"*, as the architect expressed it. Furthermore having the numbers

written down in a report made it less renegotiable at the next meeting. Also the obvious fact that you cannot require goal level 5 for every quality made the discussions more explicit about the evitable trade-offs than before using aSQA. With regards to postulate two, continuous assessment, no comments were given due to the short time the technique had been in use. Finally, the architect did find the aSQA technique lightweight once the initial hurdle of getting a common understanding of qualities (getting a common agreement by architects as well as stake holders) and levels.

On a question on the liabilities of the approach, the architect noted the problem that current and target levels are not independent due to the grading into levels 1–5. If a component's current level is evaluated to 3, it is by the aSQA definition *acceptable*. An acceptable target, however, will be interpreted from the viewpoint of the stakeholders and thus the grading introduces a large element of what the target level was supposed to express.

5.3 aSQA at Jyske Bank

The interview of the Jyske Bank architect was conducted mid December 2008. Interestingly, the architect told about an application of the technique in quite another way than it was intended for. The bank makes use of a third party banking system for their foreign countries units but the supplier had decided to discontinue the additional functionality for wealthy customers. Thus the bank has decided to buy a new such system and found two suppliers. The interviewed architect was head of the evaluation team that spent about two and a half week with technical representatives for each supplier. Previous evaluations of this type had produced various checklists but the architect found these more experience-based and less systematic than aSQA. Thus she decided to test and apply aSQA within this context rather than apply it in a pilot, in-house, development project. Architecturally, the two systems were very different, one being SOA based and open which was valued high, the other being "black-box" but cheaper and with a better graphical user interface.

This has some obvious implications for the steps in the process. First, the evaluation contains only one component that is the whole, third-party, system. Second, the current level was reinterpreted as the "supplied" level and only this and importance was considered, the latter expressing the qualities of an "ideal" supplied system as seen from the bank. The architect then evaluated the difference between current and importance to find the best. Thus the iteration aspects of the process were disregarded all together. With regards to choice of quality model and scale, they adopted those of Systematic. Regarding the outcome of the aSQA evaluation, it coincided with the architect's "gut feeling" of which was the most suitable of the two systems.

Concerning the hypothesis, the reflections were similar to those made at Bang and Olufsen. With regards to aSQA forcing quality attributes to be explicit, she replied that *"it forces you to get around and you are sure that you have turned every stone"*. No comments could of course be given regarding the *continuous evaluation* postulate. Finally, with regards to the lightweight

property, she reflected that once the ISO quality framework was acquired by the organization she would probably agree. An aspect that she too valued was the business communication aspect: providing spreadsheets to the business decision makers rather than textual recommendations adds credibility to the architectural analysis.

Finally, the architect described an idea for using aSQA as a technical benchmark when considering project proposals. Often new projects are proposed based on strong business cases but rather weak technical ones. Nevertheless the architecture department at Jyske Bank has to evaluate these based upon a one hour interview with the project leader and technical people to assess if it involves central architectural problem sets. She envisioned that aSQA could provide a better checklist during this assessment work and ultimately serve as a tool for project managers to study and evaluate the architectural aspects of the proposal.

6 A survey study

We studied the application of aSQA in the context of a continuing education course on software architecture at the University of Aarhus. The course was a 15 ECTS course (equivalent to a quarter of a year of study) in which twenty experienced software developers and architects participated while working full time. In average, the participants had 10.0 years of software development experience (median 8.5 years, standard deviation 6.2 years, least experience of 3.5 years, most experience of 25 years). The participants characterized their job roles as "developers" (18), "architects" (8), "project managers" (3), and "other" (2).

As part of a hand-in, we asked the students to apply aSQA as well as ATAM and the CBAM to a model case. ATAM is a scenario-based method for analyzing software architecture design with the purpose of identifying trade-offs and risks in design decisions. CBAM is a method for economically analyzing alternative design decisions with respects to costs and benefits.

The purpose of the study was to get participants to rate whether aSQA was applicable to their development practice given that they had knowledge of two related evaluation methods (ATAM and CBAM). We see these methods as complementary so the rating does not constitute a ranking.

The model case concerned a system for house heating control ("HS-07") for which the students had previously produced architectural descriptions based on an architectural prototype [5]. They were told to apply aSQA twice: before and after a redesign of the architecture.

The participants worked in groups (containing two to four persons) and all groups were able to apply all methods successfully. Subsequently, we asked the participants to rate whether the methods "could be useful in [their] current work" using a five-point Likert scale with the 1 being "highly disagree", 2 being "disagree", 3 being "neutral", 4 being "agree", and 5 being "highly agree". Figure 4 shows a summary of the results. 15 out of 20 "agreed" or "highly agreed" with aSQA being useful. The immediate conclusion to the survey is

that the participants were able to quickly learn aSQA (and ATAM and CBAM) and that in general aSQA scored better than both CBAM and ATAM in our rating.

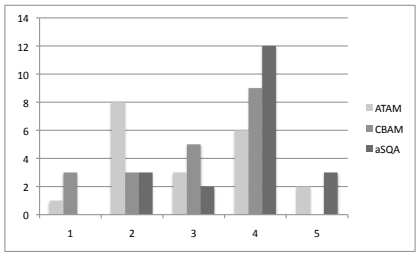

Figure 4: Summary of responses to usefulness of architectural methods. Answers were reported on a five point Likert scale.

7 Discussion

The case studies, both within and outside Systematic, as well as the survey study, generally support the original goals set for the aSQA technique, as outlined in section 3: a lightweight assessment process that quantifies quality attributes in a way that supports prioritizing. By introducing the technique, there has been improvements in terms of focus and knowledge on quality attributes in the projects. It improves communication of quality aspects between both technicians and non-technicians and makes it easier to compare and prioritize between different quality aspects. The output is a compact presentation of quality which is both absolute (as regards health) and relative (as regards focus) with respect to project goals and priorities. Health is correlated with the absolute gap between the absolute goal and the current level. Focus is relative to the current state of the project and to how much focus is currently to be invested into closing a gap between goal and current level. The technique has been in widespread use within Systematic without any major changes or refinements. Jyske Bank has since our interview tested the technique on one additional project with success and plans are presently discussed on how to roll it out in the larger context.

With regards to the reported case studies in Bang and Olufsen and Jyske Bank, the continuous evaluation aspect has not been studied and is thus not supported. The survey study involved a modest amount of iteration in that aSQA was applied twice. In the case study, however, both companies found that a primary outcome of the aSQA process can in some way be said to have nothing to do with aSQA in particular! The very premise of aSQA is to force stakeholders to evaluate and prioritize a set of quality attributes in some quality model, like ISO 9126, the framework of Bass et al., or similar frameworks. In the external case studies this process led stakeholders into heated discussions

about "what quality X really means!" If stakeholders have different perceptions of a quality (and they have) they will assign different levels. Getting to an agreement on a specific current, target, or importance level, simply force stakeholders into deep discussions on how to understand qualities and force a common understanding. This common understanding is vital in a team and important to avoid *architectural mismatch*.

Concerning the cost of applying the method, it is crucial to define an appropriate granularity of the selected components. Too fine-grained components implies that the cost of the evaluation is too high compared to the overall effect of the evaluation. Too coarse-grained components implies that it is difficult to use the information for adjusting focus on a given quality attribute during software development process in a sufficiently precise manner. The granularity also depend on the size of the system that is being developed and what the evaluation results and insights will be used for. For high-level management, coarse-grained components will often be sufficient, e.g. sub-systems of a large solution, while developers need information on a finer granularity, e.g. sub-components constituting a sub-system. In other words, the granularity of the components will depend on what level the system is managed. A solution architect who is responsible for at portfolio of systems will most likely not be interested in internal components within each system, but only consider the dominant components of each system. A system architect who is responsible for a single system (or application) will need to divide the system into relatively fine-grained components which gives sufficient information to make him able to lead the developers to focus on small parts of the system. At Systematic, ten components seems to be a reasonable number of components which keeps it manageable with respect to the cost-benefit balance.

Another aspect the software architects from all case studies reported and valued high even though it is more of a people issue than purely technical issue is the *compelling argument of numbers in spreadsheets.* Business decision makers are used to convincing arguments in the arena of economy, budgets, and project management in the form of spreadsheet accounts. Software architects primarily communicate by other means, whether it is graphical design diagrams, quality attribute scenarios, or similar. However, aSQA's output is a scoring of quality attributes and prioritizing in the form of spreadsheets like that shown in figure 3. This communication medium is simply much more similar to the language of business decision makers, and architects within Systematic report this a major benefit of the technique as their opinions simply are given much more weight when deciding resources.

On the other hand, it is necessary for the parties being involved in analysing and understanding the results to have a common agreement on the complete scope of the components. For example, a product or project being developed over several years may change its scope over time. As the scope of the project (and thereby the components) may gradually be defined it is often necessary to redefine the goals periodically in order to be aligned with the changing scope. This may lead to some confusion on what functionality is actually within the scope of the components. Essentially, it is a matter of change management –

but in practice it turns out to be difficult to grasp changing scope during quality assessments.

Even when having experience with defining components it may be difficult to get an appropriate granularity. A common problem is that the components may be so coarse-grained that the results of the analysis are of limited value in managing the quality of the component in the right direction. For example consider the following. If a large solution consisting of several modules is considered as one component then the reliability of one module is high while it is low in another module in within the same solution. When mapping this information into one number representing the current level of the solution, the ability to differ at the finer granularity disappears. This makes it impossible to use aSQA to communicate that the reliability of the other module must be improved. Therefore it is important to select the size and content of the components at a level that corresponds to what the information is to be used for at management level.

Regarding the future, Systematic is currently investigating if it is better to consider functionalities (or services) as the ones being monitored instead of the inner architectural components in the context of projects using Scrum and feature-driven development (where functionality is added as vertical slices to a system). Focusing on end-user functionalities may make it easier to communicate and discuss quality aspects with non-technical stakeholders than if the basis is the more technical components.

8 Conclusions

This paper has described a new lightweight and continuous architecture software quality assurance technique called *aSQA*. The technique is defined in terms of a seven-step, iterative, process in which software architects selects a set of components whose quality attributes are considered important to assess and control. The process is based upon selecting a quality model, like ISO/IEC 9126, and defining a uniform, course gained, metric for evaluating quality based upon stakeholder perception. Within each iteration and thus assessment, architects evaluate each component's current level for each quality attribute in question, and the aSQA model then yields a prioritization of which quality attributes for which components should receive the most attention.

The paper has outlined experiences with the aSQA technique both from within the company that invented the technique, Systematic A/S, as well as two other Danish companies. Furthermore, a survey study has been conducted. These data strongly suggests that the technique is a viable and valuable tool for software architects; that it can be made with a relatively low investment in time, especially if the selection of components and their granularity is made with care; that it support getting a common understanding between stakeholders of architectural quality issues; and that it helps in making the evitable prioritizing of which quality attributes that should receive the greatest attention.

This said, it would be highly interesting to get more data from other com-

panies and/or research institutions testing the technique. We therefore invite the scientific community to try it out and thereby explore further aspects and refinements of the technique.

9 Acknowledgments

The aSQA technique was a team effort and many people have contributed. However, the principal designers are Bo Lindstrøm, Dennis Pedersen, Henrik Kjær, and Søren Skovsen from Systematic A/S.

The research presented in this article has been funded by Systematic A/S and the ISIS Katrinebjerg competency centre, Aarhus, Denmark (`http://www.isis.alexandra.dk`). We thank the companies and software architects that participated in the SA@Work project.

References

[1] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski. Recommended best industrial practice for software architecture evaluation. *Software Engineering Institute Technical Report, CMU/SEI-96-TR-025*, 1996.

[2] H. Al-Kilidar, K. Cox, and B. Kitchenham. The use and usefulness of the ISO/IEC 9126 quality standard. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, page 7, 2005.

[3] M. Babar, I. Gorton, N. Ltd, and A. NSW. Comparison of scenario-based software architecture evaluation methods. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 600–607, 2004.

[4] M. Babar, B. Kitchenham, and R. Jeffery. Comparing distributed and face-to-face meetings for software architecture evaluation: A controlled experiment. *Empirical Software Engineering*, 13(1):39–62, 2008.

[5] J. E. Bardram, H. B. Christensen, and K. M. Hansen. Architectural Prototyping: An Approach for Grounding Architectural Design. In *Proceedings of Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA4)*, pages 15–24, Oslo, Norway, June 2004.

[6] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 2nd Ed.* Addison-Wesley, 2003.

[7] H. B. Christensen, K. M. Hansen, and K. R. Schougaard. SA@Work - A Field Study of Software Architecture and Software Quality at Work. In *Proceedings of Asia-Pacific Software Engineering Conference (APSEC) 2008*, pages 411–418, Beijing, China, Dec. 2008.

[8] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies*. Addison-Wesley, 2002.

[9] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies*. Addison-Wesley, 2006.

[10] L. Dobrica and E. Niemelä. A survey on software architecture analysis methods. *IEEE Transactions on software Engineering*, pages 638–653, 2002.

[11] ISO/IEC. *Software engineering – Product quality – Part 1: Quality model*, 2001. ISO/IEC 9126-1:2001.

[12] C. R. Jakobsen and K. A. Johnson. Mature Agile with a Twist of CMMI. *AGILE Conference*, 0:212–217, 2008.

[13] R. Kazman, L. Bass, M. Webb, and G. Abowd. SAAM: A method for analyzing the properties of software architectures. In *Proceedings of the 16th international conference on Software engineering*, pages 81–90. IEEE Computer Society Press Los Alamitos, CA, USA, 1994.

[14] R. Kazman, M. Klein, and P. Clements. ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, 2000.

[15] L. Northrop, P. Feiler, R. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, et al. Ultra-large-scale systems: The software challenge of the future. *Software Engineering Institute*, 2006.

[16] M. Poppendieck and T. Poppendieck. *Software Development: An Implementation Guide*. Addison-Wesley, 2006.

[17] J. Sutherland, C. R. Jakobsen, and K. Johnson. Scrum and CMMI Level 5: The Magic Potion for Code Warriors. *AGILE Conference*, 0:272–278, 2007.

[18] Systematic a/s web site. http://www.systematic.com/Home, 2009.

[19] L. Zhu, M. Staples, and R. Jeffery. Scaling Up Software Architecture Evaluation Processes. *Lecture Notes in Computer Science*, 5007:112, 2008.