A Summary of the Architectural Software Quality Assurance (aSQA) Method

Klaus Marius Hansen

Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark
Email: klaus.m.hansen@daimi.au.dk

Abstract. This report gives a brief introduction to the Architectural Software Quality Assurance (aSQA) method for architectural analysis. Its key characteristics are 1) a focus on quality attributes, 2) support for continuous assessment, and 3) lightweight resource demands. The method has been qualitatively evaluated in a large software development organization and in a semi-controlled experiment with seven participating groups.

1 Method Description

The architectural Software Quality Assurance (aSQA) is a software architecture evaluation method targeted for continuous evaluation of a software architecture for a system under construction with a focus on fulfillment of quality attribute requirements. It has been developed by Systematic Software Engineering and is further described in [6] and an overview of its application is given in [4]. The method can be classified as a metrics-based measuring technique according to the framework of Clements et al. [2], but it is also possible to use quality attribute scenarios [1] as a basis for evaluation.

In the following, we use a Point-Of-Sales (POS) system example to illustrate the method. The POS case is taken from Larman [5] and consists of two main parts: 1) a system to assist shop assistants (e.g., in checkout lines in supermarkets) for which software for terminals and scanners need to be written and 2) a back-end system that interfaces with external systems such as an accounting system and that implements an application layer for the terminals.

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 9126 [3] and Bass et al. [1] 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

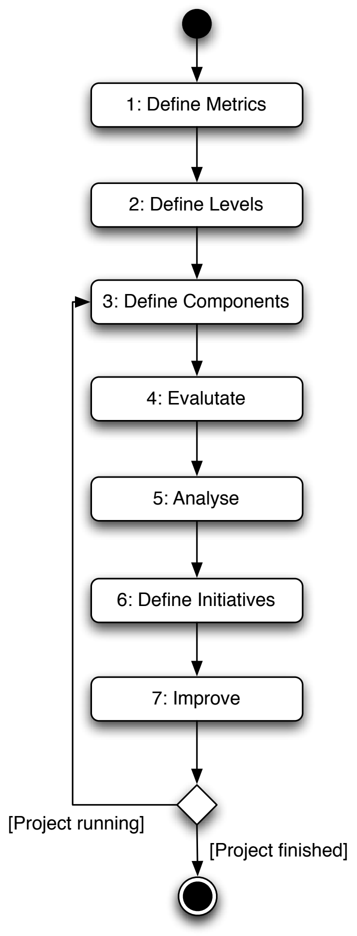

Fig. 1. aSQA Overview

and manageable comparison of values across quality attributes. An example of how to define levels is [6]:

- Level 1: Unacceptable
  - Important stakeholders find system unacceptable because of quality level of the attribute in question
- Level 3
  - No relevant stakeholder find system unacceptable because of quality level of the attribute in question
- Level 5: Excellent
  - All relevant stakeholder 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 ususally made in an iterative and possible incremental process in which the set of components may change; in the POS example, the set of components could be a Terminal, Scanner, and Application Server. The component structure should preferably be stable and it should be meaningful to assign a level of quality to that component.

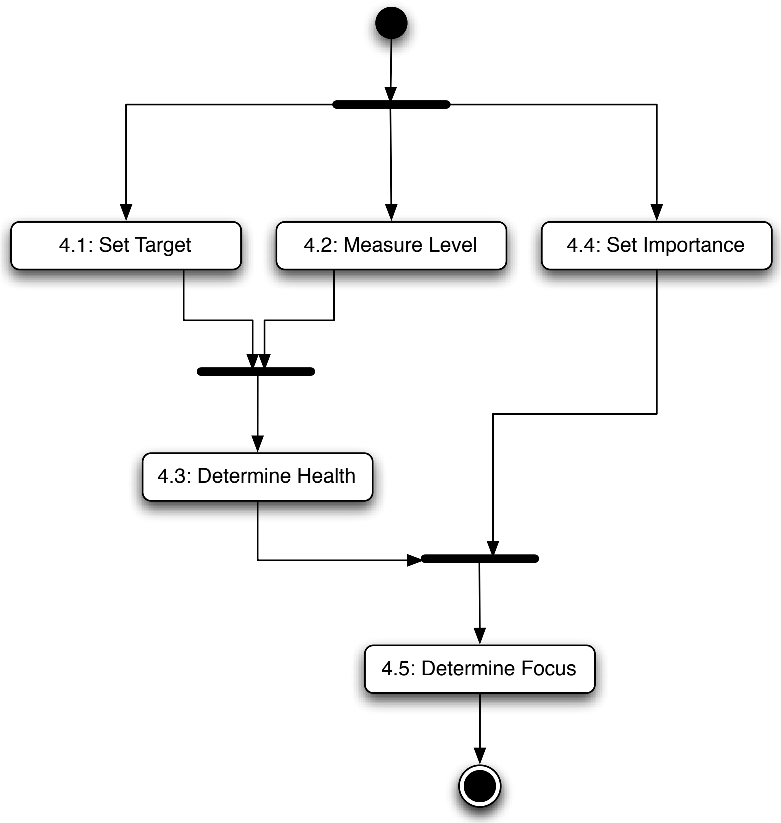

Fig. 2. aSQA Detail

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

	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

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

for the POS example. The specific calculation performed for health follows the following formula (that assigns health according to how far below the target level the measured 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 the quality attribute for the component. In the POS 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 define initiatives to improve quality (Step 6), and finally to do the improvement (Step 7).

Acknowledgments

The research presented in this paper has been partly funded by the ISIS Katrinebjerg competency center, Aarhus, Denmark (`http://www.isis.alexandra.dk`).

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* Addison-Wesley, 2 edition, 2003.
2. Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies.* Addison-Wesley, 2002.

3. ISO/IEC. Software engineering—product quality, part 1–4, 2001. ISO-9126-1,-2,,3,-4.
4. Bo Jensen and Bo Lindstrøm. Software quality in practice. http://www.daimi.au.dk/ATiSA/material/2007-11-15%20Software%20Quality-%20in%20Practice.pdf, 2007. Accessed 2008-05-16.
5. C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.* Prentice Hall PTR, 2002.
6. Bo Lindstrøm, Henrik Bærbak Christensen, and Klaus Marius Hansen. aSQA. A software architecture-level application of the ISO 9126 quality framework. In preparation, 2008.