An Integrated Quality Assurance Framework for Specifying Business Information Systems

Frank Salger[1], Stefan Sauer[2], Gregor Engels[1,2]

[1] Capgemini sd&m AG, Carl-Wery-Str. 42, D-81739 München, Germany
[2] University of Paderborn, s-lab – Software Quality Lab, D-33095 Paderborn, Germany
frank.salger@capgemini-sdm.com, sauer@s-lab.upb.de, engels@upb.de

Abstract. The software specification acts as a bridge between customers, architects, software developers and testers. If information gets lost or distorted when building this bridge, the wrong system will be built or the system will not be built in time and budget—or both! Standards and recommendations give advice on how to structure specifications or check software-engineering artefacts with reviews or inspections. But these constructive and analytical approaches are not well integrated with each other. Moreover, they are often too generic to efficiently support the specification of particular system types. In this paper, we present the integrated "specification framework" of Capgemini sd&m. It consists of our specification method for business information systems (BIS) and its concerted analytical counterpart, the "specification quality gate". Since this framework is tailored to the specification of large BIS, it allows a quick ramp-up phase for software engineering projects without the need for extensive tailoring or extension.

1. Introduction

Software quality assurance is gaining increasing attention throughout the software lifecycle. While software testing has become mainstream in professional software development, sophisticated quality assurance in the early phases of software development is still not evolved that far. In addition, it is widely recognized that international software engineering standards like IEEE 830 [5] and IEEE 1233 [6] or established software development methodologies like the Rational Unified Process (RUP) [7] are typically too generic to be directly applicable and thus need to be refined and tailored to the problem at hand (see, e.g. [9]).

We further have to acknowledge that quality assurance measures must be strongly tied and fit to the development techniques to be effective. However, a detailed companion method for performing analytical quality assurance on the software specification is typically missing, and very little work investigates the interplay between constructive specification methods and analytical quality assurance methods. For example, the approach discussed in [4] solely considers use cases. Other important artefacts such as dialogues, batches, reports or interfaces to external systems are not considered. Even RUP provides little guidance with regard to evaluating software specifications. Again, those approaches to assess a software

specification (see [1] for a comprehensive list of approaches) give little guidance for software specifications of particular system types.

From this observation, we have been developing constructive and analytic software development methodology in sync. Means of software specification, design and implementation are developed pairwise with software quality assurance techniques.

In this work, we focus on our *software specification framework* and demonstrate its strong integration of constructive and analytic software engineering methodology. We created a *specification method* particularly for business information systems (BIS) which are developed in custom projects. The method is thus immediately applicable in the BIS domain and quickly creates much value. The quality assurance tasks that are done by the specification teams are complemented by quality audits in the form of *quality gates*. We defined the quality gates in accordance to the specification method to solely assess the software specification of BIS.

The same pairing of constructive software engineering methodology and quality assurance methodology by quality gates was developed for software architecture and design [8], and will be defined for requirements specification and integration testing.

It is the tight integration of our constructive and analytical method, and the focus of the methods on one kind of system that makes the difference. It provides high benefit in terms of reduced faults, increased quality, efficient project ramp-up for these activities and high development performance. We believe that this kind of typing of constructive and analytical methods is a necessary condition for their success in practice. Thus, our specification framework for BIS contributes to the important research area of comprehensive quality assurance approaches [1].

2. Constructing Quality: The Specification Method

The Capgemini sd&m specification method has been particularly designed as a unified specification method for custom development of large-scale BIS. It looks at software systems from a conceptual viewpoint and abstracts from the technical details that are addressed during design. The final deliverable of the specification method is the *software specification*. It is also the focus of the specification quality gate.

When we specify BIS, we account for typical challenges and specifics of this kind such as the structuring of the system into conceptual components, the specification of use cases and application functions, different types of interfaces with external systems, or the structure and layout of user-interface dialogues and printed documents that are to be produced by the BIS. Therefore, the content and structure of our software specifications is tailored for the type of system we deal with and as such differs e.g. from the specification parts proposed in IEEE 830 [5].

Our specification method distinguishes the aspects *content* (i.e., specification concepts to be used), *form* (representation of concepts, use of specification languages, artefacts to be produced), *process* (specification tasks and how to proceed when specifying BIS), and *tooling*. These aspects are defined on a common basis: the ontology of the relevant concepts and a meta-model, which together constitute a domain model of software specification of BIS. The method is supported by

company-standard tool-support, provided by feeding the CASE tool (we currently use Enterprise Architect) with predefined specification templates.

The specification method is composed of *principles* and *specification modules*. The principles describe the specification method in general and cover all elements of software specifications and the process of specifying BIS. Specification modules precisely describe the major artefacts and how they are produced during specification, or groups of such artefacts (see Fig. 1). Principles and specification modules are mapped into UML profiles which are implemented in the CASE tool templates.

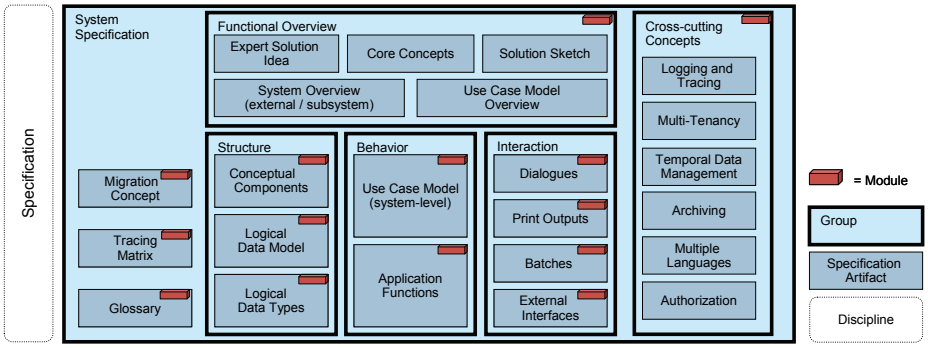

Fig. 1. The specification modules for producing the software specification

3. Analysing Quality: The Specification Quality Gate

We first derive some core requirements that we believe every comprehensive evaluation method for software specifications should possess. We then describe the specification quality gate and discuss how it fulfils these requirements.

3.1. Requirements for a Specification Evaluation Method

We have identified six core requirements that should be fulfilled by a comprehensive evaluation method for software specifications:

R1. The evaluation method is based on a quality model reflecting the interests of the different stakeholders of a software specification.

R2. The evaluation method checks whether the overall structure of the software specification constitutes an appropriate solution model for the problem at hand.

R3. The evaluation method mitigates the problem of "getting routine blinded".

R4. The evaluation method supports the evaluation of both products and processes.

R5. The evaluation method must be applicable early enough to allow for efficient correction of systematic flaws.

R6. The evaluation method fosters consistency of the software specification downstream in the development process.

We present how our quality gates fulfil these requirements in the next section. Requirements **R1** through **R5** are addressed by the specific design of the specification quality gate. Requirement **R6** is addressed by the architecture quality gate.

3.2. The Specification Quality Gate

With our *quality gates* (see [8] for the basic concept), we (1) make the maturity of development artefacts and processes transparent; (2) derive effective countermeasures for major problems encountered—this also serves to improve the constructive means; (3) "standardize" audits to a reasonable extent.

The main features of applying quality gates are: (1) They are executed according to a defined quality-gate process and end with a decision: „Am I ready for the next step?" (2) They are no formal checks but evaluate the content of artefacts. (3) They are conducted by company experts (i.e., senior architects who are not involved in the project); this addresses requirement **R3**.

The major steps in all our quality gates are planning, preparation, quality gate workshop, and reporting. In addition, the continuous improvement of the whole quality-gate process is an ongoing activity.

The specification quality gate follows the idea of perspective-based reading [2]. Different evaluation steps (given below) resemble the viewpoints of different stakeholders. Some of the evaluation steps contribute to more than one stakeholder role. Using perspective based reading as an underlying principle addresses **R1**.

The specification quality gate is targeted on specifications of BIS: For example, in its first evaluation step, we assess the compliance of the specification with respect to the meta-model of the specification method—which in turn is tuned towards the specifica
based or

Basic
overview
model. Further, about 30% of the software specification (use cases, user interfaces, etc.) has to be defined in all detail. The exact date is determined with the aid of a comprehensive list of work products expected as input to the quality gate, together with their required detail and status. It also helps the assessors to browse the software specification. We found that a software specification which already contains the use-case model and 30% of the software specification in detail is stable enough for reasonably applying the specification quality gate. Yet it is not too late for substantial correction if structural flaws should surface. This effectively addresses **R5**.

We now present the different evaluation steps that make up this quality gate.

Evaluation step 1: Structure of the software specification (addresses **R1** and **R2**)

A good structure supports the readability of the software specification. This in turn supports many stakeholders fulfilling their tasks.

How: Applying a specific questionnaire plus executing specific change scenarios.

Evaluation step 2: Systematic allocation of requirements (addresses **R1**)

Appropriate traceability is one of the main success factors of projects since it allows for effectively and efficiently dealing with change.

How: Applying a specific questionnaire plus executing specific change scenarios.

Evaluation step 3: Specification anti-patterns

A system might be perfectly specified, but if the corresponding solution cannot be built in time and budget, the project will fail. Hence, we check the software specification's feasibility.

How: Application of an internal list of known specification anti-patterns.

Evaluation step 4: Comprehensibility (addresses **R1** from customer's point of view)

One of the most difficult tradeoffs in writing a software specification is to balance readability with respect to the customer and precision of the software specification. Software engineers often tend to focus on the latter and neglect the former. This can lead to painful situations where the customer does not understand the software specification, but signs it off anyway. The discussions then emerge soon after the first running software is released: "This is not what we understood we will get!"

How: Walkthrough, navigating through major use cases from input at the user interface to output like data "logically committed", orders placed, invoices sent, etc.

Evaluation step 5: Cohesive partitioning of functionality (addresses **R2**)

In software design, the conceptual components (cf. Sect. 2) are further decomposed into logical components. If the conceptual components are of little functional cohesion, the logical components which refine them will be of little functional cohesion as well. As a consequence, the logical components have to communicate a lot with each other, leading to technical complexity. Further, they will be difficult to maintain, since many unrelated business concepts are necessary to comprehend them.

How: Application of a specific questionnaire.

Evaluation step 6: Hot spot analysis

After having applied the steps described above, the assessor has a precise understanding of the core problem areas. He can focus on these specific 'hot spots' to produce a clear description of the discovered problems. This will later serve as the basis for the definition of precise countermeasures.

Evaluation step 7: Process questionnaire

The previous steps concentrated on the evaluation of the software specification. But projects often fail due to weak processes, like inappropriate change management processes. We therefore also check the software specification processes.

Altogether, we thus defined evaluation steps that assess both the product and the processes, thereby addressing **R4**.

After the software specification passed the specification quality gate, project reality continues. At some point, we have to assess again whether the software specification is still of high quality. We do this in our architecture quality gate [8]. There, the focus is not on a complete evaluation of the software specification. Rather we evaluate samples of the software specification in order to assess whether the quality criteria ascertained in the specification quality gate still hold, and the "good practices" are still followed. In the architecture quality gate, the architecture is also evaluated against the specified quality attributes (aka. non-functional requirements) using an adapted form of the architecture trade-off analysis method [3]. This addresses **R6**.

The effectiveness of the specification quality gate's steps has been evaluated with several large projects. We did not experience that one evaluation step is more effective than the others. Different projects have different problems. However, we experienced that a key principle is that assessors are not themselves members of the project: This unbiased view proved to be extremely valuable.

4. Conclusion and Future Work

We presented the basic ideas of our overall quality assurance framework for software specifications. The specification method constitutes consolidated knowledge and best practices from twenty-five years of software engineering, enriched by the practical application of recent developments in theory and practice, on how to specify business information systems (BIS). The specification quality gate ensures that the fundamentals of the software specification are of high quality, such that

- the customer can understand the software specification,
- the software specification serves the developers and testers as a solid and manageable basis for their work,
- the software specification partitions the business problem, and the conceptual components serve as a sustainable basis for deriving the architecture and design,
- the current processes are precise, communicated and followed,
- the processes for the following activities are in place and reasonable.

The specification quality gate is being applied in real-world, large scale projects and has shown there which benefits can be obtained. Part of our future work will be the extension of the specification quality gate to our distributed-delivery model in order to support our farshore projects. The definition of quality gates for the requirements engineering discipline and the integration testing are further topics.

References

1. Aurum, A., Wohlin, C. (Eds.): Engineering and Managing Software Requirements. Springer (2005)
2. Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S., Zelkowitz, M.: The Empirical Investigation of Perspective-Based Reading. Empirical Software Engineering 1(2), 133–164. Springer (1996)
3. Clements, P. Kazman, R., Klein, M.: Evaluating Software Architectures Methods and Case Studies. Addison-Wesley (2002)
4. Denger, C., Peach, B.: An Integrated Quality Assurance Approach for Use Case Based Requirements. In: Proc. Modellierung, LNI vol. P-45, pp. 59–74, GI (2004)
5. IEEE Standard 830-1998 – Recommended Practice for Software Requirements Specifications. IEEE, New York (1998)
6. IEEE Std. 1233-1998 – Guide for Developing System Requirements Specifications. IEEE, New York (1998)
7. Rational Unified Process. IBM Corporation. Version 7.0.1 (2007)
8. Salger, F., Bennike, M., Engels, G., Lewerentz, C.: Comprehensive Architecture Evaluation and Management in Large Software-Systems. In: Proc. Quality of Software Architecture (QoSA‘08). LNCS vol. 5281, pp. 205–219. Springer (2008)
9. Wiegers, K. E.: Software Requirements. Microsoft Press (2003)