Investigating the Impact of Software Requirements Specification Quality on Project Success

Eric Knauss, Christian El Boustani, and Thomas Flohr

FG Software Engineering, Leibniz Universität Hannover
Welfengarten 1, D-30167 Hannover, Germany
{knauss,el.boustani,flohr}@se.uni-hannover.de

Abstract. Different Software Requirements Specifications (SRS) are hard to compare due to the uniqueness of the projects they were created in. Without such comparison, it is difficult to objectively determine if a project's SRS is good enough to serve as a foundation for project success. We define a quality model for SRS and derive required metrics using the Goal-Question-Metric approach. These metrics were applied in roughly 40 student's software projects. Based on this we find a quality threshold for project success. This paper contributes in three areas: Firstly, we present our quality model. It was derived from literature, and contributes to the discussion of how to objectively measure requirements quality. Secondly, we share our evaluation approach and our experiences measuring SRS quality. Others could profit, when planning to measure requirements quality. Finally, we present our findings and compare them to related studies in literature.

Keywords: Quality of Requirements, Metrics for Requirements.

1 Introduction

One of the main difficulties faced by Quality Management during the requirements analysis in software projects is to decide, whether a software requirements specification (SRS) is *good enough*. This is due to two major problems:

1. It is hard to measure *how good* a SRS is, i.e. determine the quality of a SRS in a quantifiable way.
2. If the quality of a SRS is determined, it still remains an open question, whether the value is *good enough* or not. The quality of the SRS has to be compared to other projects.

Basically, our hypothesis is that the quality of a SRS strongly influences the probability of its project success. In this paper we show that the quality can be measured mainly based on formal and objective metrics. This is important, because it allows to assess the chances of a project based on SRS quality internally. If quality is below a certain quality threshold, the project is more likely

F. Bomarius et al. (Eds.): PROFES 2009, LNBIP 32, pp. 28–42, 2009.

to fail. In order to determine this threshold we investigated roughly 40 projects based on the Goal-Question-Metric method. Based on our results we found two specific thresholds:

A lower threshold: Projects that have a SRS's quality below this value are highly endangered.
A higher threshold: Projects that have a SRS's quality above this value are likely to succeed.

These results have been discussed in [1]. In this contribution, we focus on the methodology and comparison of our results. We start with an overview of literature dealing with quality of requirement in Sect. 2. Based on this literature we decided what aspects to measure. We give examples of some promising and more sophisticated metrics in literature and discuss why they are not appropriate to our work.

The Goal-Question-Metric method basically demands for a top-down approach. Therefore, we start by stating our measurement-goal and our hypotheses. We also define, when a hypothesis is supported and when it is falsified. This is done in Sect. 3.

In Sect. 4 we explain how the measurement was done. We summarize our findings and discuss them. Finally, we compare them to the results of other empirical studies.

Next, we need to show how trustworthy our results are: We think that our empirical results are valuable to others and want to make them comparable (and hence transferable). Therefore, we discuss the validity of our results in Sect. 5.

We summarize our results in Sect. 6. We also give hints on how to enhance the reliability of similar studies.

2 Requirements Quality in Literature

Quality aspects and metrics for requirements have been widely discussed in literature. The difficulties are well known and were often discussed (e.g. see [2]). Obviously, it is very difficult to obtain objective data. For example, it is hard to determine, whether a requirement specification is complete or not. Without the original stakeholders it is impossible to decide if it contains all the requirements.

There are many textbooks [3,4,5,6] that describe how to write high-quality requirements specifications. The quality gateway in [5] is a well known example: Only *good* requirements can pass it. A *good* requirement fulfills several quality criteria. For example, only requirements are allowed to pass the quality gateway that state how to decide whether they are met. However, this process does not help to compare the requirements specifications of different projects.

Rupp [6] gives a more analytical approach to requirement's quality. Well-known quality aspects, like completeness, are revisited. But where the Robertsons [5] defined completeness based on requirements templates, [6] shows how to find incomplete requirements based on natural language. Both approaches

help to enhance the quality of SRS, but do not help to quantifiable compare it. Nevertheless, the quality aspects in these textbooks are the foundation of our work.

Davis [3] gives some suggestions on how to quantify the quality of a requirements document: Findings are weighted according to their severity. Accordingly, the overall quality of a document is the amount of findings multiplied by their weight. We integrated this suggestion as well as the proposed weights into our approach. In order to compare two SRS, we had to normalize the result by the amount of their requirements.

Besides the rather basic quality-metrics for requirements discussed above, many more sophisticated ones were suggested in science. For example, the clearness of terms is discussed in the CLEAR-method [7]. Another example is the discussion of the ambiguity of *and* respectively *or* in natural language [8]. Based on our measurement goals we had to discard these promising metrics, because they either constructively influenced the RE-process or were simply too difficult to measure.

3 Study Goals

The GQM (Goal-Question-Metric) method suggests a top-down way of goal-oriented measurement. The basic steps are defining measurement goals, describing the goals in a more detailed way using a table (the Abstraction Sheet), to formulate questions, and to derive metrics from the Abstraction Sheet that help to answer the questions in a quantifiable way (see [9]).

By filling out the Abstraction Sheets we formulated hypotheses about how good the quality goals are reached at the moment. Those hypotheses are expected measurements results. After the elicitation of data we are able to verify the hypotheses and determine if they were correct or not. We only give a sketch of our GQM-tree, examples of metrics, and our main hypotheses here.

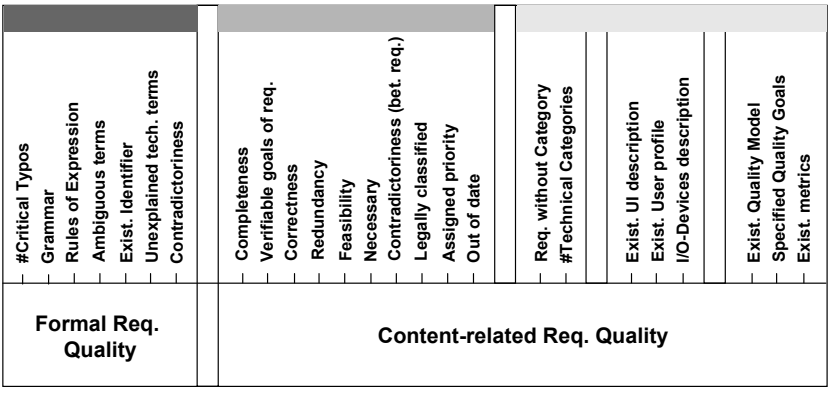

Fig. 1. Measurement goals and metrics for Requirements Quality

Figure 1 gives an overview of goals and metrics in our study. We planned to systematically measure the quality of Software Requirements Specifications (SRS). We have two subgoals: the formal requirements quality and the content related requirements quality. The main goal is to assess the quality of a SRS and to draw a connection to project success. The term *formal requirements quality* refers to verbalization rules as in [6] (e.g. completely specified process words, avoidance of incomplete comparisons, etc.). In contrast, *content related requirements quality* refers to goals that need interpretation to some extend. For example to judge, whether the SRS contains a quality model or not, the assessor has to search for quality aspects and decide, if they were sufficiently detailed. Content related requirements quality is subdivided into general, technical, UI, and quality aspects.

3.1 Project Settings

The objects of this study are software projects conducted in university teaching. These software projects (SWP) are part of the curriculum of our bachelor in computer science. All participants had basic courses in programming languages, data structures and algorithms, as well as in software engineering and project management. Participants with additional knowledge in one or more advanced courses like databases, artificial intelligence, or requirements engineering were evenly distributed among the project teams.

Each project team consists of five members. The students had to elect a project manager among themselves for their course. The projects last one term (four months) and students spent approximately 16 hours a week for their project.

We try to let our students experience a realistic software project. Therefore, each project has a customer with real interest in the final software product. This is important to determine project success. In addition, we limit the time our students may spend to interview the customer. Time for technical questions or advice is also limited.

Our students have to follow a strict waterfall development process. They start with an analysis phase, go on to design phase, and finally implement the software in the last phase, before the customer accepts (or sometimes even rejects) the software in a final test.

We measure the quality of SRS at the end of the analysis phase. At this point the requirements are frozen and the design phase builds upon them. Our motivation for this work is to identify projects that are in trouble. We run up to 9 projects in parallel. All projects reach the end of the requirements analysis at the same time. A typical SRS has more than 30 pages and contains more than 50 functional requirements. If we want to help, we need to find the project that needs our sparse resources the most.

3.2 Hypotheses

In order to judge a project's success we interviewed each customer and asked him to rate the success based on the following scale:

++ The project's results (i.e. software) are used in the intended way.

\+ The project's results could be used in the intended way, but there are better solutions available to reach the customer's goals.

\- Projects in this category failed to reach some of the customer's goals. The customer believes that these goals are reachable within a month of rework.

-- This category consists of projects that failed to deliver working software. These projects failed the acceptance tests and the customer does not believe that it would pay off to continue the project.

Note that our definition of project success differs from [10]: Our projects cannot overrun time and budget, because they are stopped at the end of term. If they cannot deliver, they have failed (category --). Only projects in category ++ are considered successful.

Concerning the relationship between SRS quality and project success we have the following hypotheses:

Hypothesis 1. Projects with a high quality-score are more likely to succeed (category ++).
Influencing factors: Relationship between formal quality aspects and quality of the SRS's content as reported in [11]. A high quality SRS might also be a sign of well-organized teams, more likely to succeed in delivering valuable software.
Hypothesis holds if we find an upper threshold with more than 75% of the projects scoring above, fall in category ++ or +.

Hypothesis 2. Projects with very low quality-score are much more likely to fail (category --).
Influencing factors: A low-quality SRS is bad enough. But teams that produce a bad SRS might have additional problems. For example, team members may work against each other or may have a bad time-managing. These difficulties may multiply as the project proceeds.
Hypothesis holds if a lower quality-threshold can be found, with more than 75% of projects scoring below, fail (- or --).

4 Conduction and Findings

This section describes, how our study was conducted. It gives an example of a metric and shows our results. We also discuss the implications for our metrics and compare our results to others.

4.1 Strategy of Measurement

Concerning the elicitation of the quantitative data we defined basic constraints. Because we wanted to compare 40 SRS and because of our limited resources we had to limit the time for the elicitation. We planned to spend less than 240 minutes per SRS.

We were interested in the relationship between formal quality aspects and project success. Therefore, we decided not to take the customers point of view into account (i.e. we did not measure whether the requirements were complete from the customer's point of view).

To enhance the speed of measurement we introduced a software tool to support the assessment of a SRS. The whole text of a SRS was copied and pasted into the tool. The tool separated each sentence and asked the assessor to decide whether it was a functional, technical, UI, or quality requirement. After that it presented each requirement and asked the assessor to look for each metric. Figure 2 shows the general process of assessing a SRS.

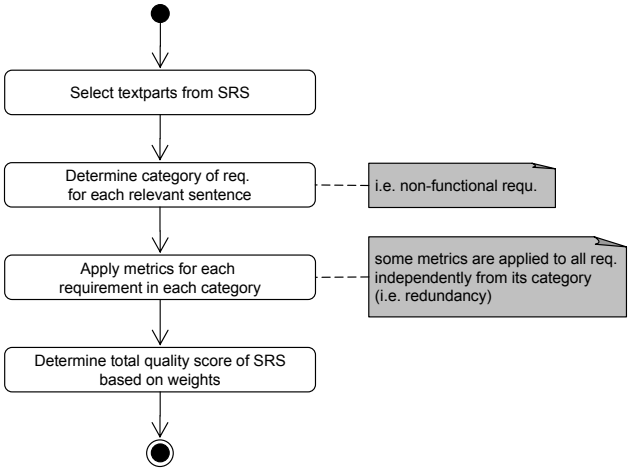

Fig. 2. Activities of analyzing a SRS

For some metrics the tool included heuristic support. For example a full-text search presented candidates for contradictory requirements. According to [12] a metric should fulfil the following requirements:

Simplicity: Effort of interpretation should be adequate. Therefore, we give the results in percentage or in numbers.

Validity: Reasonably correlation between metric and measured property. Metrics were created using the GQM-Method, the heuristic tool-support only indicates possible flaws.

Stability: Stability of the metric against manipulation of lower subordinated consideration. We consider this to be fulfilled because of using percentage and numbers.

Timeliness: Elicitation must be early enough to be able to adjust the process. This is met, because the measurement of the SRS takes place in the first stage of the project.

Analyzability: One should be able to put data of measurements in relation. Our results are given in percentage or comparable numbers.

Repeatability: Objective measurement-criteria must exist. Subjective exertion of influence must not be possible. This is only partly fulfilled because of the subjective factors in some metrics (see section 5).

Table 1 gives an example of a metric.

Table 1. Example of a metric based on 1

Metric	Formular
Verifiable goals of reqs.	$\frac{\sum verifiable\ aspects\ of\ req.}{\sum allaspects\ of\ req.}$
Simplicity:	Yes, because of percentage scale
Stability:	Yes, because of percentage scale
Timeliness:	Yes, because of measurement of the SRS in the first stage of the project
Analysable:	Yes, because percentage number can be easily compared
Repeatability:	Partly, because the decision whether or not an aspect of an requirement is verifiable, may differ from person to person

4.2 Results and Discussion

Figure 3 show an excerpt from the data we obtained. This data covers the 16 projects conducted in the last two years. The colors of the bars reflect the part of the goal-tree in figure 1:

Gray: Results of metrics that belong to the *formal requirements quality* measurement goal.

Light gray: Results of general metrics that hold for all requirements and belong to the general *content related requirements quality* measurement goal.

White: *Content related requirements quality* metrics that are specific for non-functional requirements.

In order to compare SRS of different projects, we want to determine a total score for a single SRS. To do so, we take a list from quality aspects for requirements, that is based on widespread standards [13] and pragmatic extensions [6]. For every aspect we add a weight, that indicates how important this quality aspect is estimated to be according to Davis proposals [3].

Let $m_i(srs)$ be metric i applied to all requirements of the SRS and w_i the weight of a quality-aspect from the Tab. 2, that is associated with the metric i as specified in our GQM-Model. The total score of a SRS with respect to quality of requirements is:

$$f(srs) = \sum_i w_i * m_i(srs) \tag{1}$$

Table 2. Weights of quality-aspects of requirements

Quality-aspect	Weight (w)
Correctness	10
Feasibility	10
Without contradiction	10
Up to date	9
Verifiable goal of req.	8
Comprehensibility	5
Quality of Necessary	3
Completeness	2
Unambiguousness	2
Assigned priority	1
Legally classified	1

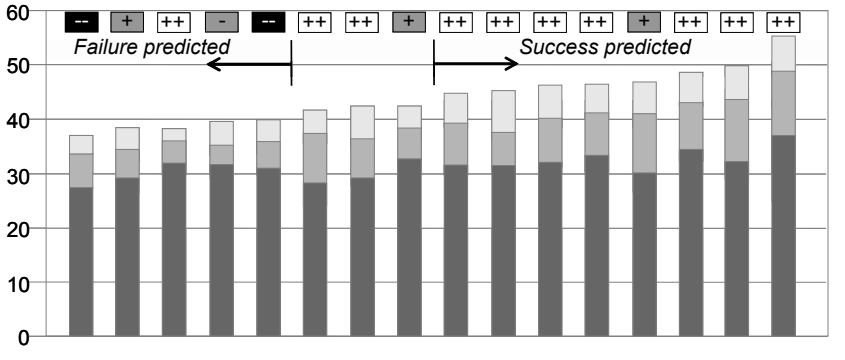

Fig. 3. Quality scores of SRS for 16 projects

Low values (below 40) indicate low SRS quality, with a high risk for errors. Values above 44 indicate high quality. Interestingly, these results generally supported each project advisor's gut feeling.

Based on our hypotheses in Sect. 3 we investigate our software projects' requirement specifications. We are interested in the dependency of a project's quality score and its success. The results support our hypotheses:

Hypothesis 1 holds: All projects that scored more than 44 Points were successful (category ++ or +). The results in Fig. 3 give even stronger support: 87% of the projects that score more than 44 on the quality-assessment fall into category ++.

Hypothesis 2 holds: All projects that failed (category - and --) scored below 40 points and we found only one project from category ++ below this threshold. Therefore, 80% of the projects below the lower threshold were not satisfactory.

Investigations of the remaining projects support this. In contradiction to the results in Fig. 3 they also show that projects that score somewhere between the upper and lower quality-threshold have a remaining risk of failure. Based on these results we do not suppose that the upper threshold could be set lower.

4.3 Comparison to Related Studies

Forsberg [14] investigated the relationship between time spent for RE and project success. Accordingly, it is advisable to spent about 20% of the time with requirements engineering. Our work cannot add to these results because our students could not decide how much time to spend for RE. All projects spent roughly 20% of their time for requirements analysis. Because of this timeboxing, we are able to draw a connection between SRS quality and project success.

In [15] So and Berry investigated how adjustments of the RE-process improved requirements engineering. This work compared two releases of a large software product. The encouraging result is that RE efforts pay-off. The evidence is based on the decreased number of bug reports and change requests. Again, we cannot directly add to these results, because our development process does not include change management and we do not track bugs after release. Therefore, our definition of project success differs too much. However we can add evidence that good requirements engineering increases customer satisfaction.

Olsson et al. [16] investigated the relationship of functional requirements and non-functional requirements (NFR) at Sony-Ericsson. Accordingly, about 40 % of the requirements specified were non-functional. This result is very interesting for our work: It shows a quantifiable (i.e. comparable) difference (despite project size) between our students' projects and high-quality industrial software projects. Table 3 shows the relationship derived from our data. The difference is much lower than expected. However, the non-functional requirements in our projects were poorly specified (e.g. there were no testable quality requirements).

Table 3. Relationship of functional requirements and non-functional requirements

Percentage of technical requirements	10%
Percentage of UI requirements	2,8%
Percentage of quality requirements	17,2%
Percentage of non-functional requirements	30%

Kamata and Tamai [17] investigate the relationship between the quality of a SRS and project success. In difference to our approach they rely upon data derived from normal quality assurance activities. Quality Agents rate each section of a SRS based on 100 criterions. Their approach of measuring requirements quality seems to rely on subjective criterions and the Quality Agent's experience. Based on the maturity of the organization, the evaluaton results can be considered repeatable.

The quality of a SRS is computed by mapping the ratings to requirement artifacts (i.e. subsections proposed for SRS in [13]). This allows Kamata and Tamai to identify critical sections for project success. In addition, computing a SRS quality profile based on a accepted SRS structure enhances the comparability of their approach. In order to compare our results to the results of Kamata and Tamai, we performed several steps:

1. *Describe fundamental differences of the approaches.* In contrast to [17], we focused on mostly formal metrics. We tried to make our metrics as objective as possible instead of relying on experienced assessors. In addition, we measured the quality as a whole. The main result is the same: Both our work reports that a correlation between SRS quality and project success exists. Table 4 shows how the different definitions of project success can be compared.
2. *Map our results to the proposed structure.* In order to compare the results we normalized the results of each metric to values between 0 and 1. Some of our content-related metrics can be mapped to the proposed structure. The others measure quality of the specification as a whole. So we filtered each finding of such a metric by the section of our specification where it was found. Then, we mapped the sections of our specification to the proposed structure. Finally, we can give a rating between 0 and 5 for each section of the proposed structure for our projects.

Based on this mapping, we can compare the profiles of our successful and failed projects to the corresponding profiles reported in [17]. Our goal was to investigate if we could identify critical sections in our projects, too. As shown in Fig. 4 we did not achieve good results. One reason for this is that our metrics do not measure quality in each section in a fair way:

Table 4. Comparing Definitions of Project success

Over Time	- / +	−−
In Time	++	+ -
	In Costs	Over Costs

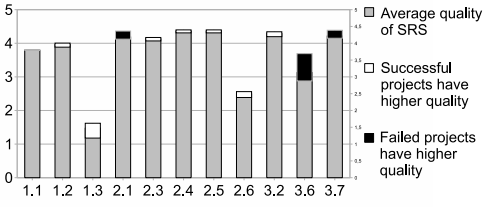

Fig. 4. Quality of our SRS per section mapped to IEEE template [13]

We have many metrics that apply to sections 1.3 and 3.2 but only our general metrics for formal requirements quality apply to section 3.7. Based on our observations we would support Kamata and Tamai in that not all sections are equally important for project success. However, our GQM-study cannot contribute to these results, because it was focussed on other measurement goals.

5 Evaluation of Validity

This section presents the threats to validity we identified during our work (according to Wohlin et al. [18]) in order to let others decide if our work is relevant for them.

5.1 Construction Validity

Construction Validity describes issues that are caused by the construction of the empirical evaluation.

The construction of our study is influenced by the design of the practical programming course we investigated. Apart from our study, students should learn to accomplish a whole software project. Therefore, we were not allowed to exchange the SRS between the groups investigated. This leads to the issue that our model might measure the performance of the students groups instead of the project success. Good students achieve good results (i.e. good SRS, good design, and finally good projects). The investigation of this issue remains future work.

5.2 Conclusion Validity

Conclusion validity mainly concerns the possibility to draw statistically significant conclusions from the empirical study.

The main problem we see here is the reliability of measures. Since some measures rely on human judgement to some degree a certain bias is probable. This problem was mitigated by conducting the measurements by the same person. To apply the measurements in a company (where measurements will be conducted by different persons) we suggest to develop measurement guidelines over time to reduce the influence of human judgement. We take a closer look on repeatability in section 5.5.

A minor threat concerns the statistical power. 16 specifications were intensely reviewed by one person. The considerable number of the remaining 24 specifications were reviewed by different persons. Our original measurement results seem to hold for these 24 specifications, too.

5.3 Internal Validity

Internal validity concerns the influences on independent variables beyond the knowledge of the reviewer.

Since all measurements were conducted by one single person measures could be applied in a different way over time, because the person got more familiar with the measurement process. The second problem concerns the fatigue. There is a considerable probability that the person got tired over time, affecting measurements in a untraceable way. This problem is considered to be low, because only one SRS was measured per day.

In addition, other factors during the later phases of the projects might have influenced the empirical results. This issue is moderated by the fact that all

observed projects followed the same process. Furthermore, we carefully observed the groups and did a post-mortem analysis, but could not observe any such factors.

5.4 External Validity

External validity concerns the ability to draw conclusions beyond the scope of our empirical study e.g. transfer our results to industry.

A rather superficial threat to external validity is that all investigated projects were conducted in a university setting. Consequently, transferability of the results poses a problem. The fact that all our projects have a common background setting helps to achieve a better result concerning data quality (which strengthens conclusion validity). Industry strength projects could be assessed in the same way and compared to our findings, because the application of our metrics is not limited in any way. We opted for conclusion validity instead of external validity in order to gain more data points.

5.5 Discussion of Repeatability

A more severe threat concerns the repeatability: How do different assessors affect the result? We let four different persons assess the same project. The results of one person differed drastically from the others as shown in figure 5.

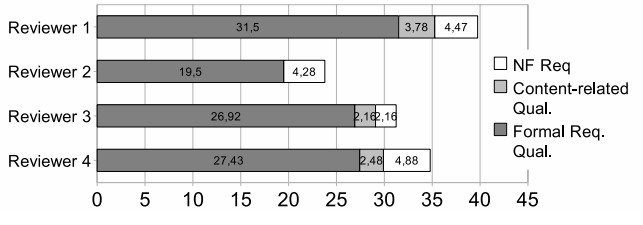

Fig. 5. Assessment of repeatability shows room for improvement

We analyzed each variably classified requirement. The main reasons for the derivation were:

1. False positives, e.g. requirement was classified as passive voice but was formulated in active voice.
2. Scope of interpretation, e.g. finding process words.
3. Generally subjective measures, e.g. understandability, redundancy, and technical terms

The effect of false positives and scope of interpretation can be reduced by more extensive training, keyword-lists, and heuristic tools. Furthermore, these errors are systematic: if one person assesses a project twice, the result will be more

or less the same. Therefore, we expect that the ranking of the projects will be more or less the same, despite different quality-scores. Bad projects will be rated worse than good projects.

Another problem is the complexity and long duration of the assessment. An assessment of a typical SRS took between 120 and 180 minutes. In addition, some of the measurements are rather complex, like finding inconsistencies between different requirements.

Based on these threats to validity there are also some issues concerning the mapping between SRS quality and project success. Because of the different scores the reference project got depended on the reviewer, there is no absolute threshold for the quality-score. However, with all reviewers creating a consistent ranking of the projects, we can give a relative quality thresholds.

Table 5 shows types of measurement faults we identified. The first column gives the type and the second column a typical example. The third column shows how we would address this problems in future work.

Table 5. Overview of Measurement Faults

Type	Example	Mitigation Strategy
1 False Positives	*found* passive voice *but was* active voice	• training • keyword-lists
2 Scope of Interpretation	*identification of process words*	• heuristic tool support
3 Generally subjective measures	*understandability*	• quantify and limit the effects

6 Conclusion and Outlook

In this work we had two goals. On the one hand we wanted to measure the quality of software requirements specifications. On the other hand we wanted to depict project risk based on this quality.

In order to do this, we had to investigate a set of software projects. Based on the GQM-method we could support our hypotheses that the quality of a requirement specification influences the probability of project success. In our setting we were even able to give a threshold: projects that are worse than this value are very likely to fail.

With this assessment of the SRS' quality we have a powerful instrument for our software projects: Based on the results we can decide whether a project is allowed to go on or whether its SRS needs major rework. Based on our assessment we also observed that the overall quality of our projects' SRS increased since last year. Such observations are essential for software organizations that want to improve themselves, because the effects of process improvements become visible.

As a side-effect we found out that even simple verbalization policies (e.g. the requirement template [6] or the use case template [19]) strongly improved the requirement quality. Such policies simply let fewer room for errors.

However, our approach has some known limitations. Our assessment cannot replace the validation of requirements. There is still the need of customer collaboration, e.g. aspects regarding the content cannot be quantified using the SRS only. Nevertheless, our assessment helps finding hazardous points in the SRS, that can be addressed during validation. Therefore, it helps increasing the efficiency of Reviews. The main drawback of our work is the limited repeatability. In section 5 we argued that this does not limit the validity of our results. However, this causes our specific threshold to be worthless for others, leaving each organization with the need to find and calibrate their own threshold.

For this reason we plan to enhance the elicitation of our metrics. On the one hand we will improve the preparation courses of our quality agents. In addition we will provide them with more detailed instructions on how to interpret a given metric. On the other hand we want to introduce more heuristic tool support. With the reduced cognitive load of separating false positives from true ones, we hope to enhance repeatability as well as elicitation speed.

Finally, we were able to compare our teaching projects to industry projects. Despite being considerably smaller and at some points not as good as their real-world siblings, we saw that our projects are highly comparable. Because of this we expect others being able to build upon our results.

References

1. Knauss, E., El Boustani, C.: Assessing the Quality of Software Requirements Specifications. In: Proceedings of 16th International Requirements Engineering Conference, Barcelona, Spain, pp. 341–342 (2008)
2. Costello, R.J., Liu, D.B.: Metrics for requirements engineering. In: Selected papers of the sixth annual Oregon workshop on Software metrics, New York, USA, pp. 39–63 (1995)
3. Davis, A.M.: Just Enough Requirements Management: Where Software Development meets Marketing (2005)
4. Gause, D.C., Weinberg, G.M.: Exploring Requirements: Quality Before Design (1989)
5. Robertson, S., Robertson, J.: Mastering the Requirements Process (1999)
6. Rupp, C.: Requirements-Engineering und -Management: professionelle, iterative Anforderungsanalyse für die Praxis (2004)
7. Wasson, K.S.: A Case Study in Systematic Improvement of Language for Requirements. In: Proceedings of the 14th IEEE International Requirements Engineering Conference, Minneapolis, USA, pp. 6–15 (2006)
8. Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying Nocuous Ambiguities in Natural Language Requirements. In: Proceedings of the 14th IEEE International Requirements Engineering Conference, Minneapolis, USA, pp. 56–65 (2006)
9. van Solingen, R., Berghout, E.: The Goal/Question/Metric Method: a practical guide for quality improvement of software development (1999)
10. The Standish Group: CHAOS Chronicles v3.0. Technical report (2003)
11. Wilson, W.M., Rosenberg, L.H., Hyatt, L.E.: Automated analysis of requirement specifications. In: ICSE 1997: Proceedings of the 19th international conference on Software engineering, New York, USA, pp. 161–171 (1997)

12. Liggesmeyer, P.: Software-Qualität. Testen, Analysieren und Verifizieren von Software (2002)
13. IEEE: IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998 (1998)
14. Forsberg, K., Mooz, H.: System Engineering Overview. In: Thayer, R.H., Dorfman, M., Davis, A.M. (eds.) Software Requirements Engineering, Los Alamitos CA, pp. 44–72 (1997)
15. So, J., Berry, D.M.: Experiences of Requirements Engineering for Two Consecutive Versions of a Product at VLSC. In: RE 2006: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE 2006), Washington, DC, USA, pp. 216–221 (2006)
16. Olsson, T., Svensson, R.B., Regnell, B.: Non-functional requirements metrics in practice - an empirical document analysis. In: Proceedings of Workshop on Measuring Requirements for Project and Product Success, in conjunction with the IWSM-Mensura Conference (2007)
17. Kamata, M.I., Tamai, T.: How Does Requirements Quality Relate to Project Success or Failure? In: Proceedings of 15th International Requirements Engineering Conference, Delhi, India, pp. 69–78 (2007)
18. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C.: Experimentation In Software Engineering: An Introduction, 1st edn. (1999)
19. Cockburn, A.: Writing Effective Use Cases (2000)