Measuring and Improving of Testability for Testing Requirements in an Industrial Context by Applying the Goal Question Metric Approach

Armin Beer
Beer-Test Consulting
Baden bei Wien
Austria
armin.beer@bva.at

Michael Felderer
University of Innsbruck, Austria
Blekinge Institute of Technology, Sweden
michael.felderer@uibk.ac.at

ABSTRACT

There are two basic constraints in testing: cost and quality. The cost depends on efficiency of testing activities and quality and testability. The practical experience of the author in large-scale systems shows that if requirements are adapted iteratively or the architecture is altered, the testability decreases. However, what is often lacking is a root cause analysis of testability degradations and the introduction of improvement measures during the development of a software. In order to introduce agile practices in the rigid strategy of the V-model good testability of software artifacts is vital. So, testability is also the bridgehead towards agility. In this paper we report on a case study where we measure and improve testability based on the Goal Question Metric Approach.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

System testing; requirements-based testing; testability; complexity; requirements quality; traceability; software quality; empirical study.

*Produces the permission block, and copyright information
†The full version of the author's guide is available as acmart.pdf document
1It is a datatype.

1 INTRODUCTION

Requirements-based testing has been recognized as the key to aligning business value and risks in industry. Testing of systems in the social insurance domain follows the V-model. The test cases are developed from the artifacts of the requirements specification. For instance, in system testing use-case descriptions, business rules and a specification of the user interface are used. Because of the long duration of projects and frequent changes of the requirements test cases have to be changed. In order to introduce agile practices in the rigid strategy of the V-model good testability of software artifacts has to be in place.

In this paper, we report about a case study for applying a methodology to manage testability in large scale projects taking frequent requirement changes into account. Based on the analysis of complexity, release planning, effort, time, defect data and experience, we performed a retrospective analysis and show how early investments in testability can improve later results. Improvements are feasible in terms of (1) reduction of complexity of requirements artifacts, (2) balancing of development and test effort (3) better final quality.

The paper is organized in the following way. Section 2 describes the terms and the background on testability. Section 3 describes the study design. Section 4 presents the background and the research questions. Section 5 presents the cases and its assessment. Section 6 covers the metrics and the evaluation of the cases. Section 7 reports the results. Section 8 closes with conclusion and future work.

2 BACKGROUND AND RELATED WORK

2.1 Testability

Software testability [1] is the degree to which a software artifact supports testing. If the testability of the software is high, then findings faults in the system by means of testing is easier. A lower degree of testability results in increased test effort [8]. In Figure 1 the relationship between testability and complexity is presented. The complexity and quality of the software requirements specification (SRS) has an impact on the effort of the design of test cases. If the complexity of the SRS is high, the testability and the maintainability of test cases are low. Traceability between the requirements artefacts for instance requirements, use cases and test cases have to be in place in order to monitor the status and progress of the project. For a good maintainability of test cases testers have to understand the relationship between the different artifacts and the impact of a change for instance of legacy requirements.

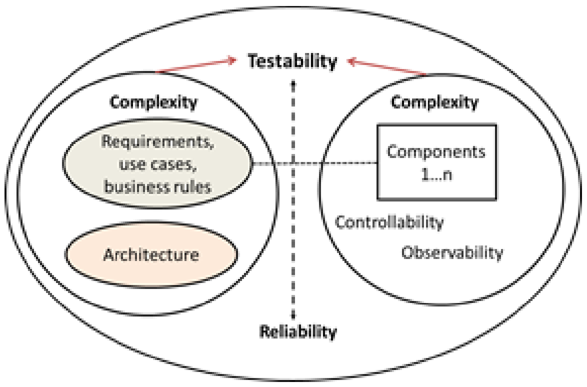


Figure 1: Testability of software systems

2.2 Complexity

A system is classified as complex, if its design is unsuitable for the application of exhaustive simulation and test, and therefore its behavior cannot be verified by exhaustive testing [5]. The IEEE Standard Computer Dictionary defines complexity as the degree to which a system or component has a design or implementation that is difficult to understand and verify [6]. This phrase suggests that complexity is relative to the observer. That means for example that a skilled analyst would analyze and design a well-structured system. Uncontrolled complexity in software may have the consequence of higher cost, less rigorous testing or reduced performance (see also NASA report [7]).

Egyed [14] lists categories of complexities, which have to be taken into account, when requirements are changed:

- Many-to-many mappings for instance requirements to design elements.
- Incompleteness and inconsistencies
- Different stakeholders in charge of different software artifacts
- Increasingly rapid change of pace
- Non-linear increase in the number of software artifacts during the course of the SW-life cycle (n^2 complexity).

Kan [9] presents complexity metrics in order to provide clues for software engineers in order to improve the quality of their work. Cyclomatic complexity (McCabe 1976) was designed in order to indicate program testability and maintainability. If an organization can establish a significant correlation between complexity and defect level, then the McCabe index can be useful to help to:

- identify complex parts of code needing detailed inspection
- identify non-complex parts likely to have a low defect rate
- estimate programming and testing effort.

Structural metrics, “Fan-in” and “Fan-out” try to take into account the interactions between modules in a system. Modules that have a large “Fan-in” and “Fan-out” indicate a poor design.

Henry and Selig [9] defined an information flow metric, combining a module complexity metrics with the structural complexity. Card and Glass [9] developed a system complexity model, which is a sum of structural (inter-module) complexity and overall data (intra-module) complexity. They found, that the system complexity measure was significantly correlated with a subjective quality assessment by a development manager and with development error rate. They provide also guidelines to achieve a low complexity design. The high correlation between module changes and enhancements illustrate the fact that the more changes, the more chances for injecting defects. Small changes are especially error-prone. Kan [9] concludes, that the key to achieving quality is to reduce the complexity of software design and implementation in a given problem domain.

2.3 Reliability

Deficiencies in the testability of requirements or components influence the reliability of a product. Grottke and Dussa-Zieger [10] relates test case coverage and the number of failures experienced. One goal of the project was to develop a SW growth reliability model.

FMEA (Failure Mode and Effect Analysis) is a bottom-up analysis to identify potential failure modes with its causes. FMEA especially takes the architecture and the complexity of components into account. In software failure mode and effect analysis (SFMEA) the complexity is taken into account to assess a Risk-Priority Number (RPN) and to define the test strength [13].

2.4 Technical debt

The technical debt approach fosters the recognition and mitigation of deficiencies of the software development process.

Alves et alt. [11] identified the following forms of technical debt:

- Architectural debt: high complexity of the overall system architecture.
- Defect debt: for instance defects not fixed, for lack of resources or low prioritization.
- Design debt; code that violates good design practices.
- Requirements debt: for instance bad testability of requirements
- Test automation debt: for instance bad testability of the system under test

2.5 Goal-Question-Metric (GQM)

To improve the software development process improvement goals have to be defined. These improvement goals should support business objectives and take the actual status of ongoing projects into account. The Goal Question Metric method of Basili [17] and van Solingen [20] provide an efficient frame work for the improvement of development and testing activities and its approval for instance by a project manager.

The GQM method contains the phases planning, definition, data collection and interpretation. The definition phase identifies a goal for instance the improvement of testability, questions related metrics and assessments. During the data collection phase collection forms for instance EXCEL templates are defined and project data for instance of the defect tracking tool are filled in. During the interpretation phase the measurements are used to answer the stated questions. These answers are again used to see if the stated goals for instance the improvement of the release quality are achieved.

The technical debt metaphor fosters an analysis of the reasons of the problems for instance the decrease of testability encountered from release to release. GQM is a powerful method to measure and mitigate technical debt.

3 STUDY DESIGN

This section presents the design of the study. The core content of our research is to investigate the effects of software testability with special focus on requirements based testing from a retrospective perspective. We evaluate two projects quantitatively and qualitatively and document the lessons learned.

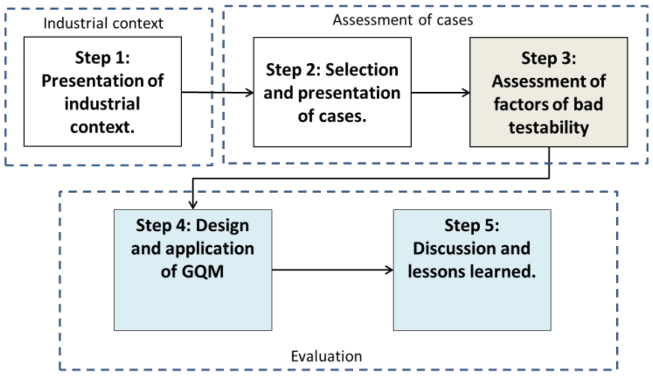

Industrial context
Assessment of cases
Step 1: Presentation of industrial context.
Step 2: Selection and presentation of cases.
Step 3: Assessment of factors of bad testability
Step 4: Design and application of GQM
Step 5: Discussion and lessons learned.
Evaluation

Figure 2: Design of the study

Step 1: We present the industrial context taking testability problems into account.
Step 2: We present the selected projects of a social insurance institution, which are assessed in the next step.
Step 3: Testability degradations are observed in the mentioned cases. We assess the reasons for bad testability.
Step 4: We define metrics to monitor the test process and perform a retrospective evaluation of two projects, applying the GQM (Goal Question Metric).
Step 5: We discuss the results and improvement measures in order to enhance testability and reduce complexity of software artifacts.

4 INDUSTRIAL CONTEXT: STEP 1

For practitioners in industry good testability of software artifacts is a key issue in order to deliver applications of high quality in time and budget. In the studied cases, testability is influenced by a growing complexity of the software artifacts for example by change requests, new interfaces, platform upgrades etc. In the case study presented by Felderer and Ramler [15] one of the major complexity drivers were additional modules introducing further inter-modules dependencies. Jungmayr [16] presents an approach to define metrics of software dependencies, ACD (Average Component Dependency) taking the extent of the influence of the dependees in software design into account.

The selected cases were analyzed in respect to find reasons of bad testability and their improvement by applying GQM.

5 SELECTION AND ASSESSMENT OF CASES: STEP 2 and 3.

Step 2 –Selection and presentation of cases

In this section, the cases selected for this study are presented.

Case A:

- A new complex system for the management of subscriptions of insurants connected with software components of different institutions of the government, companies and the bank are under development.
- Frequent requirements changes during development have an impact on testability
- Development of the software lasts about 5 years and the total effort is about 10.000 person days.
- Complex performance requirements
- Synchronization with release plans of coupled systems.

The degradation of testability is recognized as a key problem when progressing in the iterative development.

C**ase C**:

- It is a core application dealing with charging of money obligations of insurants and the healthcare institution
- Analysts created requirements and component of good testability at the start of the project.
- New interfaces to external components were added during development.

Attributes	**Case A**	**Case C**
Goal	Management of subscription of social insurants.	Automation of business of social insurance.
Duration of project [year]	4	7
Number iterations	10	4
Number of releases	25	8
Number of features	65	20
Number of req.	370	80
Iteration 3 Test effort plan [PD]	217	215
Percentage of test effort to development effort	40%	30%
% of manual TCs to be automated	50%	50%

Figure 3: Description of Cases

Step 3 – Assessment of factors of bad testability in the social insurance institution

The factors effecting testability encountered in projects of a social insurance institution, we used the testability fishbone visualization of Binder [19] and transformed it in a mind map (Figure 4). We will focus rather on process issues, than on technical problems.

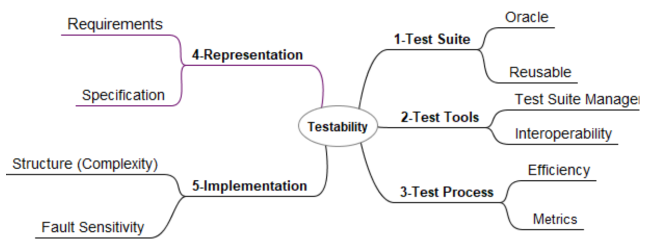


Figure 4: Testability factors in SW-projects

1-Test suite: The test oracle is deduced from the requirement specification. Traceability should be in place in order to adapt test cases when requirements are altered. Good testability is a prerequisite for an efficient design and execution of test cases.

2-Test tools: In system testing tools should support test suite management, test case development and reporting. To embrace changeability, tools for analysis and testing should be tightly coupled

3-Test process: The underlying model of the test process is the V-model. The effort and duration of verification and validation phases lead to iterations of 3-4 months each. The amount of detected bugs influences the degree of testability. Metrics to monitor the test process are missing or difficult to evaluate. High effort of maintenance of automated test diminishes return of investment of automation.

4-Representations: Usefulness and testability of representations is a critical factor of test-case design. In our case natural language and semi-formal representations in UML are used. The specifications should contain relevant information for system testers on domain level as well as detailed technical descriptions for developers.

5-Implementation: The system architecture can be too complex: Structural (Fan-in and fan-out) and system complexity (inter-module and overall data complexity) are not taken into account. The detectability and localization of faults and the effect of a failure may be difficult. Permanent performance issues may hinder the development and the execution of automated regression tests of the iterations.

6 EVALUATION: STEP 4: QUANTITATIVE AND QUALITATIVE EVALUATION OF CASES

To prevent degradation of testability during the development cycle we need a framework and metrics. We apply the goal-question-metric (GQM) approach of Basili [17] and van Solingen [20]. First goals, questions and metrics are defined. Measurement data are collected and questions are answered in order to improve the efficiency of testing.

6.1 Planning phase: Definition of goals, questions and metrics

The primary **goal** for the test management are to keep testing efficient, to reduce the time needed for testing and to improve the quality achieved by testing. This goal can be refined to more specific goals in order to improve testability:

G1. Requirement artifacts should be testable and support the viewpoint of developers and testers.

G2: The quality of the releases or iterations should continually improve.

G3: Change requests should have a low impact on architecture and test cases.

G4: The effort of localization, correction and retest of software defects should be minimized.

G5: The ROI of test automation should increase during the testing of releases.

6.1 Planning phase: Definition of goals, questions and metrics

G1. Requirement artifacts should be testable and support the viewpoint of developers and testers.

Q1: What is the testability and complexity of the requirements to be tested?

M1.1: C*omplexity of features and use cases*
M1.2: Degree of testability
M1.3: Categories of requirement anomalies, i.e. the requirement anomalies detected by a review

G2: The quality of the releases or iterations should continually improve

How can we assess the quality of releases and the efficiency of test results?
M1.1 Test results and report
M1.2- Defect trend analysis.

G3: Change requests should have a low impact on architecture and test cases.

Q1: What are the consequences of change requests to testing? To which extent we have to take the dependee components into account?
M1.1 Number of regression test cases
M1.2- Test effort

G4: The effort of localization, correction and retest of software defects should be minimized.

Q1: How can we assess the testability of the architecture.

M1.1 Testability of architecture and average component dependency

M2.1 Effort to analyze, fix and retest defects.

G5: The ROI of test automation should increase during the testing of releases

Q1: How can we measure the degree of automation of test cases in order to reduce the time of regression tests?

M1.1 Number and percentage of automated test cases.

6.2 Interpretation.

The consultant is a member of the test management group the social insurance institution and has access to the project data for instance project plans, effort estimations, minutes of meetings, defect tracking etc. Questions relevant to the requirements, development and test were discussed with the stakeholders. The data were collected in spreadsheets and checked by the test manager and the leading analyst. Goal attainment, answers and measurement are evaluated in cases A and C. In order to compare the testability of case A with case C we will focus on iteration 3 of these cases. The functionality of Case A and case C has to be compatible and both iterations are the final before shipment.

The goal of the measurement was to understand root causes of the increase of testability problems from iteration to iteration and to find remedies in order to meet the deadline of shipment of the applications. A second goal was to define a measurement program for future projects.

We will now present the application the GQM method to the cases.

G1. Requirement artifacts should be testable and support the viewpoint of developers and testers.

Q1: What is the testability and complexity of the requirements to be tested?

M1.1: Complexity *of features and use cases*

The degree of complexity is assessed by the analyst and the system architect. The skill of teams and the testability of features or requirements are taken into account. The effort of analysis, development, architecture and test can be calculated more precisely by taking these influence factors into account. The relation of the factors assigned to analysis, development etc. as depicted in Figure 5 are heuristic.

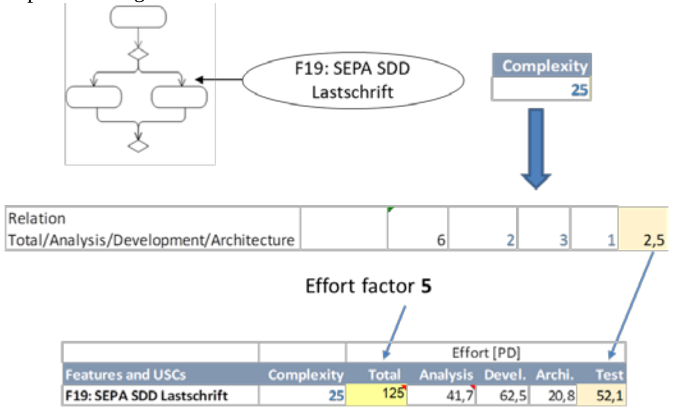


Figure 5: Case C: Calculation of complexity and effort of development

In case A the complexity of requirements artifacts is assessed by assigning attributes (low and high) only.

Complexity measures allow a realistic calculation of the effort of analysis, development, architecture and test. This conclusion was based on the experience in case C.

M1.2: Degree of testability

The number of TC's not executed and the percentage of automation, indicate the severity of testability problems encountered. As depicted in Figure 6 only 17% of TCs could be automated in iteration 3. In case C about 37% of TCs where automated. Feedback of domain experts, developers and testers of case C indicated also, that the requirement specification is of good quality i.e. testability and understandability. [3]

Release	Passed	Failed	Not exec.	Number of manual and automat. TCs	Number of automat. TCs	% automated
Case-C Rel. V 3.8.4	1.424	111	109	1.644	607	36,92%
Case A Rel. V 0.8.0	1.909	292	393	2.594	441	17,00%
Case-C Rel. V 3.8.5	880	45	706	1.631	603	36,97%
Case-A Rel. V 0.9.0	1.390	178	947	2.515	457	18,17%

Figure 6: Degree of testability in case A and C

In case A the amount of change requests cumulated in iteration 3. It showed the importance to build in testability already in the first iteration.

M1.3: Categories of requirement anomalies

Case A: The test management assessed the quality of requirement specification of iteration one and detected the categories of defects depicted in Figure 7. All testability issues had a high impact on the design of test cases.

Testability issue	Description	Example
Completeness	Use-case descriptions	Links in use case description are missing.
Defects	Business rules	Rules contradicting
Traceability	Mapping	Coverage of business processes
Comprehen-sibility	Natural language	Focus on the developer point of view, too technical for domain experts.
Right level of detail	For developers and tester	Sorting rules of items are not detailed enough to design test cases.

Figure 7: Categories of anomalies of SRS detected by reviewers in iteration 1

A great effort is invested in the review of requirements. The example of case A is depicted in the following table. The anomalies detected by the different reviewers. Major and critical anomalies detected, were testability problems from the viewpoint of testers (Figure 8). Testability issues detected by testers were regarded as problems to design good test cases. [3]

Case A	Review		
Iteration	Number of Reviewer	Number of anomalies	Category 2 (major) oder 3 (Critical)
2c	11	269	4
3b	11	557	2
3d	7	641	10
3	10	219	8

Figure 8: Number of anomalies of SRS detected by reviewers

G2: The quality of the releases or iterations should continually improve

Q1: How can we assess the quality of releases?

M1.1 Test results

Case A: In Rel. 0.6.0 1782 test cases were executed, about 20 % failed, because of requirement changes of an external component and schedule 30% of the planned TCs couldn't be executed. In Rel. 0.7.5 the database had to be redesigned and also the test cases had to be changed. Insufficient performance and interface defects were the reason that 30% of the detected defects had been "critical". This result indicates, that the more changes the more defects will occur [9].

In Rel 0.8.0: despite of "feature freeze" the number of open bugs still increased, because the localization of defects was difficult and the testability was bad. The development resources were insufficient to correct the amount of bugs in time.

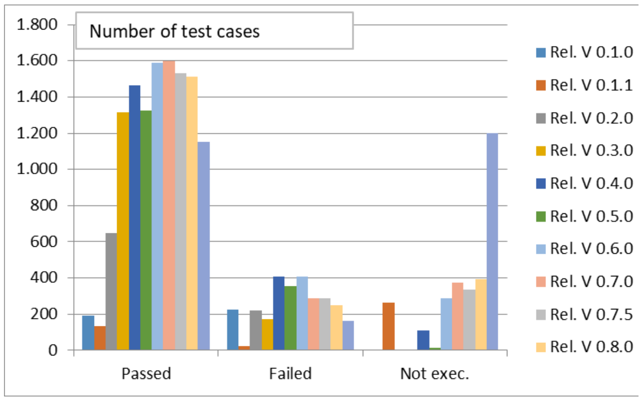


Figure 9: Case A: Result of system tests of releases

Case A: Figure 9 depicts the test progress starting with the first iteration. The number of not executed TCs increased in the last iteration before shipment because of still latent testability issues.

M1.2- Defect trend analysis.

The defect analysis of case A shows that the total amount of open bugs (not fixed or not retested) still increased when testing iteration 3. The number of open bugs, i.e. not fixed or not retested, is still increasing.

G3: Change requests should have a low impact on architecture and test cases.

Q1: What are the consequences of change requests to testing? To which extent we have to take the dependee components into account?

One major challenge is the synchronization of the release plans of case A and case C during the last iteration before shipment. For example had the backlog of not corrected defects and performance problems an impact on testing in case C.

M1.1 Number of regression test cases

Testability problems cumulated in Case A Rel. V 0.8.0, because of the following issues:

- Complexity of the requirement specification increased compared to iteration 1 and 2.
- Nearly the complete set of test cases had to be changed because of numerous change requests
- 28% of TCs could not be retested because of testability problems after bug fixing
- 35% of test cases failed because of testability problems

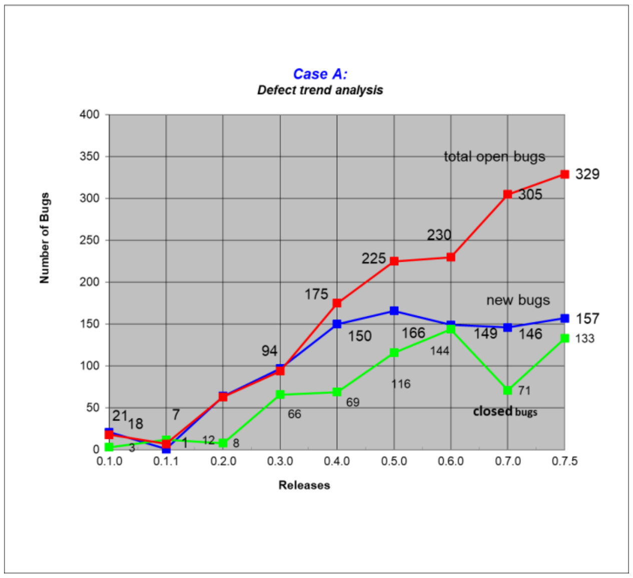
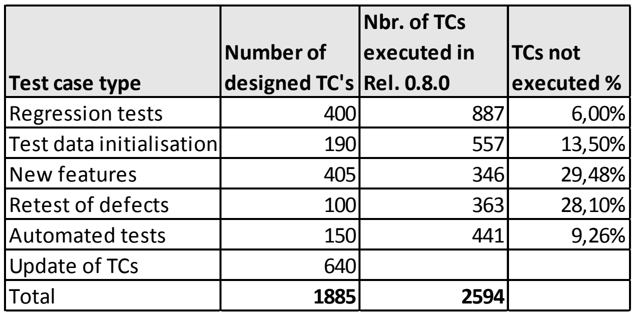


Figure 10: Defect trend analysis of Case A



Test case type	Number of designed TC's	Nbr. of TCs executed in Rel. 0.8.0	TCs not executed %
Regression tests	400	887	6,00%
Test data initialisation	190	557	13,50%
New features	405	346	29,48%
Retest of defects	100	363	28,10%
Automated tests	150	441	9,26%
Update of TCs	640		
Total	**1885**	**2594**	

Figure 11: Percentage of TCs not executed in Case A Rel. V 0.8.0

M1.2- Test effort

The test effort is estimated taking the testability of the requirements into account and differs from project to project. For the design of test cases in case A 1.5 hours in Case C 1.25 hours per test case are estimated. The execution of a manual test case is calculated with 0.5 hours per test case. The test automation is performed by a separate team and the effort is calculated separately.

G4: The effort of localization, correction and retest of software defects should be minimized.

Q1: How can we assess the testability of the architecture.

M1.1 Testability of architecture and average component dependency

In case C the architecture took potential changes of interfaces or the connection to new components into account. Component dependencies are low.

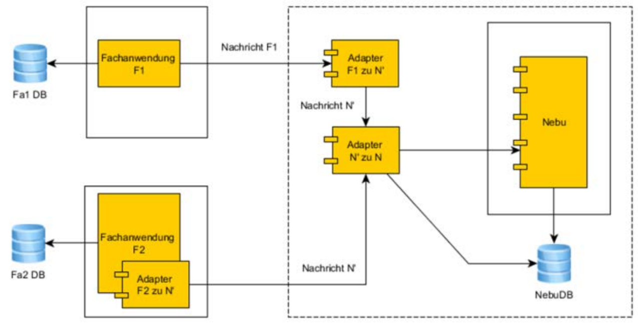


Figure 12: Case C. testable architecture

M2.1 Effort to analyze, fix and retest one defect

Case A: The localization, correction and test of defects is labor intensive and takes according to the experience of the test manager about one day per defect. The main reasons are: bad testability of the software under test, high complexity of architecture (many interconnected components) and the complex relations between distributed data sets, which are changed from iteration to iteration.

G5: The ROI of test automation should increase during the testing of releases

Q1: How can we measure the degree of automation of test cases in order to reduce the time of regression tests?

M1.1 Number and percentage of automated test cases.

Case A: The goal to automate 50% of the test cases could not be implemented in case A, because of the frequent changes of the use cases and masks. Only 17% of the planned automated test cases were automated in iteration 3.

Iteration	Case A rel.	Total nbr. of TCs	% automa-ted
	Version 0.1.0	420	
	Version 0.1.1	420	no
	Version 0.2.0	868	
1	Version 0.3.0	1.489	
	Version 0.4.0	2.117	6,38
	Version 0.5.0	1.779	4,72
2	Version 0.6.0c	2.493	8,46
	Version 0.7.0	2.258	no
	Version 0.7.5	2.154	
	Case A Version 0.8.0	2.594	17,00
3	Case A Version 0.9.0	3.020	16,66

Figure 13: Percentage of executed automated TCs in Case A

6.2 Data collection and threats of validity..

The validity of this conclusion concerns data collection and the reliability of the measurements, any of which might affect the ability to draw the right conclusion. The source of the data are test effort estimations, review results, minutes of meetings, protocols of steering committee, test reports, the test case design and execution documented in a test management tool etc.. The data of the study, are collected in spread sheets as well as a questionnaire and were checked by the authors and the test manager of the social insurance institution.

7 EVALUATION: STEP 5: DISCUSSION AND LESSONS LEARNED

In the interpretation phase we will draw conclusions regarding the results of the study. The results are discussed in feedback sessions with the test manager. Presentation slides are prepared to focus on the main findings. The study yields that the test manager should be involved during the analysis and design phase with the goal to reduce complexity and enhance testability to mitigate not foreseen changes in implementation and testing of coming change. Metrics should be used to monitor and foster a continuous improvement of the procedures of development and testing. We will now discuss improvement measures taking the interpretation of the results of the application of the GQM into account:

(1) Reduction of complexity and enhancement of testability:
Requirements or platforms are changed, new interfaces are added in an iterative development process. As a consequence, the complexity of the software artifacts increases and testability is worsening. In black-box testing the observability of subsystems has to be fulfilled in order to implement effective test cases. However, in case A subsystem cannot be accessed to verify test results. In case A the relatively simple top-level requirements hide the immense complexity introduced by legacy and dependency on components of a networked system. Therefore analysts should follow guidelines in order to mitigate complexity and focus on testability, For example in case C, analysts added examples of abstract test cases are added to business rules and enhanced its testability by sign pointing changes with colors. In case C the testability improvements, introduced by the analyst early in the analysis phase are accepted by the domain expert, testers and the project management. The comparison of the percentage of the

automatic regression tests of adjacent releases of case A and case C, depicted in Figure 6, indicate a better testability in case C. Due to testability problems for instance delays in test case design and defects not corrected, the amount of untested test cases in case A, as depicted in *Figure 9*, are high.

(3) Working packages of testers

Working packages assigned to testers should be elaborated in time and budget. However in case A with testers of skills [2] comparable to case C the completion of working packages are delayed by lack of understandability of requirements and observability of the software under test. In case A the effort of fixing and retesting of defects has been one day per defect, because the localization of a fault needed a frequent communication between developer and tester. For the estimation of effort and duration of working packages the degree of testability has to be taken into account. An assessment of the complexity, as depicted in Figure 5 improves the definition of WPs.

(4) Frequent change requests

Change requests may have an impact on a great amount of test cases. Therefore traceability from a test case to the requirement artifacts has to be in place. Tools of analysts and test management should be tightly coupled. Alternate release plans should be taken into account [18].

(5) Efficient test management

Metrics presented in chapter VI should be taken into account to monitor a test project. Bug fixing and retesting should have priority.

(6) Shorter iteration and release cycles

In order to get an earlier feedback the current duration of the implementation of a release of about 6 to 12 months should be reduced.

(7) Good testability of the system architecture

The long duration of projects and the networking of different systems is a challenge for the system architect. Testability of architecture, components and data has to be taken into account at the start of the project. The analyst of case C has implemented a testable architecture taking the coupling of new components into account.

8 CONCLUSIONS

To sum up, our study indicates that by applying the GQM method specific quality attributes can be monitored in order to create a framework for the improvement of testability. Experience in two large scale projects under development shows: Testability is the key for an efficient development of a software product. In future, we will perform further empirical studies and refine our framework for measuring testability based on the Goal Question Metric approach.

ACKNOWLEDGMENTS

The paper was partly funded by the Knowledge Foundation (KKS) of Sweden through the project 20130085: Testing of Critical System Characteristics (TOCSYC).

REFERENCES

[1] ISTQB. "Standard glossary of terms used in software testing", Version 3.1, Internat. Software Testing Qualifications Board, Glossary Working Party, 2016.
[2] A. Beer, R. Ramler, "The role of experience in software testing practice", Proceedings of the 34th Euromicro Conf. Softw. Eng. and Advanced Applications: 258-265, IEEE 2008.
[3] M. Felderer, A. Beer, B.Peischl, "On the role of defect taxonomy types for testing requirements: Results of a Controlled Experiment", Proceedings of the 40th Euromicro Conf. Softw. Eng. and Advanced Applications, pp. 377-384, IEEE 2014.
[4] H. Femmer et al., "Rapid requirements checks with requirement smells", Journal of Systems and Software, Elsevier 2017.
[5] Defense Standard 00-54, Requirements for safety related electronic hardware in defense equipment, UK Ministry of Defense, 1999.
[6] IEEE Standard Computer Dictionary, 1990.
[7] Dvorak D.L., Eds., NASA study on flight software complexity, Jet Propulsion Laboratory, California Institute of Technology, 2001.
[8] Wikipedia, retrieval Dec.28, 2018
[9] Kan St., Metrics and models in software quality engineering, Addison-Wesley, pp. 311-330, 2006.
[10] Grottke M., Dussa-Zieger K.; Prediction of software failures based on systematic testing, EuroSTAR, Stockholm, 2001.
[11] Alves,N. S. R., Ribeiro L.F., Caires V., Mendes T. S., Spnola R. O., Towards an ontology of terms of technical debt. Proc. IEEE 6th Int. Workshop on Managing Technical Debt, pp 1-7, 2014.
[12] IEEE Standard Computer Dictionary, 1990.
[13] Sulaman M.S., et al, Comparison of the FMEA and STPA safety analysis methods – a case study, Softw. Quality Journal, Springer, 2017.
[14] Egyed A.; Tailoring Software Traceability to value-based needs. In Biffl St. et alt. (Eds.): Value-based software engineering, pp.287-308. Springer, 2010.
[15] Felderer M., Ramler R., A multiple case-study on risk-based testing in industry, Int. Journal on Software Tools for Technology Transfer 16(5), Springer pp 609-625, 2014
[16] Jungmayr S., Testability measurement and software dependencies, Proceedings of the 12th International Workshop on Software Measurement. Vol. 25. No. 9. 2002
[17] Basili V.R., Rombach H.D., "The TAME project: Towards improvement-oriented software environments", IEEE Trans. On SW Engineering, vol.SE-11, pp 758-773, 1988.
[18] Felderer M., Beer A., Ho J., Ruhe G., Industrial evaluation oft he impact of quality driven release planning, ESEM '14, Torino, Italy, 2014
[19] Binder R.V., Design for testability in object-oriented systems, Comm. Of the ACM, Sept. 1994, Vol.37, No.9, pp 89-101, 1994.
[20] Van Solingen R, Berghout E., The goal/question metric method: a practical guide for quality improvement of software development; McGraw Hill, London, 1999.