Multi-Dimensional Measures for Test Case Quality

Christian Pfaller, Stefan Wagner
Technische Universität München
Software & Systems Engineering
85478 Garching, Germany
pfaller@in.tum.de
wagnerst@in.tum.de

Jörg Gericke, Matthias Wiemann
Siemens AG
Corporate Research & Technologies, CT PP 6
81739 Munich, Germany
joerg.gericke@siemens.com
matthias.wiemann.ext@siemens.com

Abstract

Choosing the right *test cases is an important task in software development due to high costs of software testing as well as the significance of software failures. Therefore, evaluating the quality of test techniques and test suites may help improving test results. Benchmarking has been successfully applied to various domains such as database performance. However, the difficulty in benchmarking test case quality is to find suitable measures. In this paper, a multi-dimensional measuring of test case quality is proposed. It has been shown that not only the number of detected faults but also other aspects such as development artefacts like source code or usage profiles are important. Consequences of this multi-dimensional measure on creating a test benchmark are described.*

1 Introduction

Testing software systems is expensive and can be time-consuming for some types of systems. Hence, the evaluation of different test techniques and test suites has been an early identified problem. Choosing the right techniques and prioritising the right test cases gives economic benefits. Therefore, it is important to analyse test case quality [9]. To this end, the number of revealed faults and its efficiency, i.e. the time spent to reveal a fault, have mainly been used [3]. Moreover, coverage measures are exploited as a direct measure of test case quality [2, 1]. Although, there are indications that there is a correlation between coverage and fault detection, we argue that considering only the number of faults is not sufficient.

Problem As we discussed in [6], the number of faults as well as coverage measures alone are not sufficient to evaluate test cases. Those measures miss very important issues such as the failure rate and safety-criticality of failures as well as their relevance to the user.

Contribution We discuss the mandatory properties of suitable measures for test case quality and propose that only a multi-dimensional measure can fulfil these needs. Especially, we stress the dependency of test case quality measures on the available artefacts, such as requirements specifications or test results. Thus, not only the test suites need to be taken into account for testing benchmarks but also the environment of the test: development process, problem domain, test technique and other artefacts.

Outline Section 2 discusses three aspects of test case quality that need to be considered when defining measures. Section 3 describes derived consequences for test benchmarks. We close with conclusions in section 4. Related work is cited where appropriate.

2 Three aspects of test case quality

Measuring quality is always a daunting task: The term ”quality” has to be defined first. Here, quality refers to the appropriateness of test cases to check the quality of a system which in turn is difficult to analyse. Therefore, the discussion focuses on three different aspects of test case quality: First, we argue that it should strongly be differentiated what kind of object is in the focus of measures, secondly, we structure different dimensions of test case quality by means of the available artefacts and documents in different phases of the development process and finally we distinguish relative and absolute measurements.

2.1 The object of measurement

Considering the variety of test quality measures, it can be observed that the object of measurement may vary. Thus, the object that is in the focus of the quality statement should be precisely determined:

- A *single* test case
- A *set* of test cases (test suite)
- A *method* which creates test suites

A quality statement about single test cases is hardly needed, especially since the sum of the quality assessments of single test cases may differ from the quality assessment for the complete test suite. A whole test suite, which is a set of test cases, is probably the more interesting object. Since a set of test cases could also comprise only one element, single test cases are covered as well.

Methods that create or generate test suites are much more difficult to measure. Like in [4], this problem is often compared to the problem of benchmarking in other fields of software engineering like performance benchmarks to analyse computer architectures or database system. We think the problem for test quality benchmarks is slightly different: In performance benchmarks, the main problem is to execute real-world problems as an experiment (e.g. it may be too hard to produce realistic loads of queries like in the real-world scenario in which a few thousands users interact with the database system at the same time). Thus, only small experiments are measured. The difficultly lies in extrapolating the results to the real world. There, benchmarks are often simplifications of the real world. The challenge is to simplify but not to over-simplify.

In contrast to that, it is not a conceptual problem to generate test cases for known real-world software system (with known faults) and measure the amount of uncovered faults. The problem in measuring test case quality is the generalisation of the results to other systems. Thus, the challenge to simplify weighs less than to find a useful representative that holds for all possible systems or development projects the benchmark could be applied to. This could also imply that a set of systems from different domains with different characteristics is needed for a general statement.

Thus, the challenge in measuring test quality is to ensure that a test method is able to find a certain number of faults in *any* development project it is applied to. If one can state that a test method reveals a significant number of bugs in a certain system, then the reason for this statement may be twofold:

- Certain *classes of faults* occur in a significant higher concentration in the tested system.
- The generation method produces test cases, which are appropriate to detect faults of this class.

Thus, a proof, that a test case generation method M' that performs better than method M in project P will also outperform in any other project P', must cover the following two parts:

(a) Describe the general characteristics C and C' of faults detected by M or M' respectively.

(b) Show that faults of characteristics C' are more likely to occur than faults of characteristics C in any project P'.

Whereas (a) might be hard to do in many cases (b) is probably impossible to show in most project situations. Only if a certain similarity between different projects could be identified, which fortifies a similar distribution of faults in the projects, one might have an idea.

2.2 Measuring against what and when?

Of course, it is desired to measure test cases by the ratio of found and not found faults. Unfortunately the actual number of faults found by a set of test cases can only be stated *after* the execution of the test cases. The number of undetected faults is not known before the end of the system life cycle. At this point, the quality of test cases is hardly of any interest, i.e. to improve the process in future projects.

As long as there is no certainty that the applied test method will perform similar on other (future) projects, this measure is of limited use for selecting a test method. A large-scale set of benchmark systems for testing could of course increase the confidence in a certain test method, but it remains very hard to proof that results in benchmarks will be equivalent for any system (cf. section 2.1). Furthermore, it needs a lot of effort to collect data for meaningful benchmarks that way. Thus, it would be reasonable not only to focus on fault detection capabilities but also on further quality dimensions: Besides the quality statements in terms of fault (non-)detection, other quality measures are often considered, for example the ever-again mentioned code coverage criteria. Reflection of users' requirements in test cases may be another quality aspect. The advantage of such alternative measurements is that they can be applied in early project phases, especially before executing the test cases. Such measures may be applied to judge whether a test suite seems suitable for the system and can be used to guide the definition of suitable test cases.

It is notable that most reasonable means for measuring test case quality do not consider test cases solely to assess some quality level, but express quality in relation to other documents, such as source code or requirements documents. Thus, the shape of measures will highly depend on the selected documents that are taken into account besides the test cases. Noteworthy documents may be for example the following:

- *Requirements specifications*, which state single requirements the system must fulfil
- *Source code* of the system implementation
- *Test results*, which show the detected failures of a test set
- *Usage profiles*, which state the likelihood that a function will actually be executed by a user
- *FMEA and risk analysis*, which identify critical parts in the system
- *Failure reports*, which show failures due to undetected faults during the systems operation

Probably some more documents may be taken into account. The availability of these documents usually depends on the development phase. Therefore, different measures may be used in different development phases. Of course a measure may also use a combination of documents, as shown in table 1.

It defines exemplarily 12 situations in which a measure may be of interest—of course, test cases are considered in any measure. Situation 1 takes only test cases into account. The number and length of test cases as well as distribution of input events may be a possible measure. In situation 2 test cases are compared with the specification, whereas in situation 3 test cases are compared to source code, which leads toward the classical code coverage criteria. In this way, for any combination of documents possible measures may be found. Considering the six mentioned types of additional documents, theoretically up to $2^6 = 64$ different classes of measures could be distinguished. Quality measures in one class are probably easy to compare or to set in relation to each other. Measures of different classes will in contrast be hard to compare because these may focus on very different quality aspects. Besides structuring comparable kinds of measures such a classification will also help to select a suitable measure since selection of a measure type will also depend on the available documents as well as on the desired expressiveness and the usage of the measures.

In [5], the missing link between adequacy criteria and fault detection capabilities is stated. If benchmark environments would exist that do not only focus on fault detection but also on the some other adequacy criteria and quality measures like described before, then these missing links between different adequacy criteria, test case quality and fault detection could become more evident.

2.3 Relative vs. absolute quality

As last thought, it should be evaluated whether it is required to have an absolute measurement. Of course, measures which allow to state that a certain set of test cases is *bad* or *good* seem advantageous but it seems very unlikely that absolute and general measures can ever be found. They would enable absolute judgements and be applicable in project phases when test cases should be defined. So far, the more realistic way of measurement is to focus on relative measures, which state that the test suite A *is better than* test suite B.

A way that seems to allow absolute judgements is an economic analysis [7]. Monetary units promise an absolute measure of value and thereby quality of tests. However, such monetary values have also several problems when comparing different test results [8]. For example, inflation and currency conversion render comparisons difficult.

3 Consequences on test benchmarks

To fulfil a benchmark up to a certain level—e. g. some test method was able to detect x percent of the known faults of the benchmark system—can be regarded as a measure itself. Thus, the requirements on measures can be applied to benchmark sets as well. In section 2.1 we noted that results of one system might be very difficult to generalise. Benchmarks as well as measures should therefore provide answers to the following questions:

- What fault situations are contained in the benchmark system (or covered by the measures)?
- What are the characteristics of the faults?
- In what kind of systems and in which project settings are these specific fault situations of an above average interest?

With the last point we want to express that it is required to analyse the system and project on which a test method should be applied to become aware if the benchmark results of a method are meaningful with respect to the project. In systems of different application domains, like embedded software or business applications, typical faults are likely to be different. The same argument holds for the project settings: The size and skills of the development team, the quality of requirements documents, the development process model and so on may influence the characteristics of typical faults in the system.

We need to answer the mentioned three questions to be able to judge whether a certain benchmark is a reliable tool to select a test method for a certain development project. For most benchmarks, detailed answers to the stated questions—especially the last one—would probably be hard to find. Thus, it may be useful to take other dimensions of quality into account as well and not only focus on ”ratio of detected faults” in a benchmark.

With the analysis of test quality measurements according to available documents as we did in section 2.2, we

Situation		1	2	3	4	5	6	7	8	9	10	11	12
Test Cases		X	X	X	X	X	X	X	X	X	X	X	X
vs.	Requirements Specification		X		X						X		
	Source Code			X	X							X	
	Test Results					X						X	X
	Usage Profile						X			X	X		
	FMEA / Risk Analysis							X		X			
	Failure Reports								X				X

Table 1. Combinations of documents used in test case quality measures

could also classify test quality benchmarks. It may be a first step towards more meaningful benchmark sets if a benchmark would also give information about requirement and code coverage, testing according to risk analyses etc. Of course, a benchmark set would not only need to evaluate the test system and the list of known defects but also respective other documents and artefacts as mentioned in section 2.2. A classification of measures and benchmarks according to the relevant documents would help to select an appropriate benchmark, which is used to select a test method. Benchmarks that consider fault detection (what they usually should do) could help to judge whether certain test adequacy criteria are actually linked to fault detection: For example, if we have a benchmark that also provides means to measure code coverage, then we are able to compare the score in this code coverage with the score in fault detection. Since a popular benchmark would probably be used by many test engineers, developers or researchers in this field, a further analysis on the relation between some measures and fault detection might be possible.

4 Conclusions and future steps

In this paper we discussed several aspects, which we regarded as important for a structured analysis of test quality measurement. We think such an analysis is helpful for defining meaningful software testing benchmarks. First, it is important to specify the object under measurement, such as a test suite (a set of test cases) or a test method. Second, we should not only focus on detected faults but also on other quality dimensions, which are given by the relation of test cases to other documents and artefacts in the development process. Finally, we should think about absolute measurement after reliable relative measures have been found.

In the future, useful measures that are be suitable to assess test quality must be chosen or defined. We could classify these measures with the scheme introduced in section 2.2. Hence, also benchmarks could be classified according to this scheme. We think this would help to select benchmarks for specific quality aspects and would enable a more easy comparison of different analyses of test methods depending on what benchmark was used and what measures they support. On the long term, such an analysis could contribute to a better understanding of the relation between different quality measures. It may help to identify the measures that have a high potential to give a reliable statement about the test quality in terms of not only fault detection. Of course, such measures must be applied in early steps of the development process.

References

[1] P. Frankl and O. Iakounenko. Further empirical studies of test effectiveness. In *Proc. 6th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE'98)*, pages 153–162. ACM Press, 1998.

[2] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. 16th International Conference on Software Engineering (ICSE'94)*, pages 191–200. IEEE Computer Society Press, 1994.

[3] N. Juristo, A. Moreno, and S. Vegas. Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, 9:7–44, 2004.

[4] J. Miller, M. Roper, M. Wood, and A. Brooks. Towards a benchmark for the evaluation of software testing techniques. *Information and Software Technology*, 37(1):5–13, 1995.

[5] M. Roper. Software testing–searching for the missing link. *Information and Software Technology*, 41(14):991–994, 1999.

[6] S. Wagner. Efficiency analysis of defect-detection techniques. Technical Report TUM-I0413, Institut für Informatik, Technische Universität München, 2004.

[7] S. Wagner. A model and sensitivity analysis of the quality economics of defect-detection techniques. In *Proc. International Symposium on Software Testing and Analysis (ISSTA '06)*, pages 73–83. ACM Press, 2006.

[8] S. Wagner. Using economics as basis for modelling and evaluating software quality. In *Proc. First International Workshop on the Economics of Software and Computation (ESC-1)*. IEEE Computer Society Press, 2007.

[9] T. Yamaura. How to design practical test cases. *IEEE Software*, 15(6):30–36, 1998.