Test case quality as perceived in Sweden

Anders Adlemo
Jönköping University - School of Engineering
Jönköping, Sweden
anders.adlemo@ju.se

He Tan
Jönköping University - School of Engineering
Jönköping, Sweden
he.tan@ju.se

Vladimir Tarasov
Jönköping University - School of Engineering
Jönköping, Sweden
vladimir.tarasov@ju.se

ABSTRACT

In order to reach an acceptable level of quality in a software product, testing of the software is paramount. Testing can be requirement-driven or test-driven, which is translated into requirement-driven test case development or test-driven test case development. The focus of the study in this paper is on the former. But no matter what type of test case development implemented, to reach "good" software quality it is necessary to rely on "good" test cases. To define the criteria that make up for a good test case is not an easy task. Some intents have been presented over the years but none of the publications have studied the individual ranking of these criteria. This paper presents the results from a study undertaken in Sweden that indicate how experts in the Swedish software industry think about good test cases and rank a set of test case criteria.

KEYWORDS

Good software test cases, quality criteria, requirement-driven testing, test-driven testing

ACM Reference Format:
Anders Adlemo, He Tan, and Vladimir Tarasov. 2018. Test case quality as perceived in Sweden. In *Proceedings of Requirements Engineering and Testing, ACM RET workshop (RET 2018).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The study presented in this paper forms part of a bigger research project named OSTAG (Ontology-based Software Test cAse Generation). The aim of the project has been to develop models for the (semi-)automatic production of software test cases based on requirements written in natural language. Some of the results from the OSTAG project can be found in [6], [7], [8], [10]. One challenge when automatically producing test cases is the ability to determine the quality of the test cases. This is the background to why the research team has undertaken the study presented in this paper.

As long as software code is written by humans, bugs will undoubtedly find their way into the code, to a greater or lesser degree. The way to handle these errors is to undertake some kind of testing of the code. An important part in testing is made up of test cases whose main attribute is to detect the aforementioned bugs. The way to procure "good" software is thus to procure "good" test cases. The problem is, though, what defines a "good" test case? As programming is considered an art, something that is mastered over time and not learned from a book, a description of what constitutes a good software test case would be of utmost importance to the software community in general, and to test developers and test executioners in specific. The contribution of this paper is to present a study that, in part, has based its input parameters on previous definitions of good test case criteria. The study includes some preliminary results of what Swedish experts in the software testing domain consider to be criteria of good test cases, including a ranking between these.

RET 2018, June 2018, Göteborg, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

The definition of what constitutes a "good" test case is not something easily captured. This observation could be one of the reasons why not too many publications can be found on the topic. The first intent to define what good test cases really entail was undertaken by Kaner in 2003 [4]. Already in the abstract he stressed that the difficulty to write good software test cases depends on the complexity to construct them. The complexity, according to Kaner, comes from three sources:

- Different types of tests are more effective for different classes of information.
- No test case will be good for all good test case criteria.
- Good domain tests are different from good, risk-based tests.

But before even trying to define criteria for good test cases, it is necessary to define what a test case really is. The ISO-IEC-IEEE 24765-2010 standard [1] states that a test case is

- a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation. 3.1.31.
- documentation specifying inputs, predicted results, and a set of execution conditions for a test item. IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation. 3.1.31.

Another defacto standard is Test Maturity Model Integration (TMMi) release 1.0 [9], that states that a test case is

- a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

As can be observed, both standards are very similar. Accordingly to Kaner, a test case is a question that you ask of the program. The

point of running a test is to obtain information, for example whether the program will pass or fail the test. But the main objective, that everyone in software industry seem to agree upon, is for a test case to detect as many software bugs as possible in the tested code.

Kaner presents some criteria for what he believes a good test case should do:

(1) Find defects
(2) Maximize bug count
(3) Block premature product releases
(4) Help managers make ship / no-ship decisions
(5) Minimize technical support costs
(6) Assess conformance to specification
(7) Conform to regulations
(8) Minimize safety-related lawsuit risk
(9) Find safe scenarios for use of the product
(10) Assess quality
(11) Verify correctness of the product
(12) Assure quality

As can be observed from the list, the criteria, on an individual level, are very different in scope, ranging from concrete activities, such as to *find as many bugs as possible*, to more elusive goals, such as to *minimize the safety-related lawsuit risk*. Such a diversity among criteria is sure to have a detrimental impact on testing. Another weakness with Kaner's work is that he does not present any individual ranking among the criteria, but this is not uncommon in this line of research.

Atmadja et al. propose a more limited list of only four criteria [2] but, again, without any ranking. Accordingly to the article, a good test case should:

(13) Have a clear purpose
(14) Focus on testing only a few aspects of the test subject[1]
(15) Produce output which can be easily verified by the tester
(16) Give reproducible errors

An important aspect that can be observed in the the list by Kaner is that some of the criteria are close to impossible to measure. For example, the criteria *find safe scenarios for use of the product*, is difficult to measure or to even assign a metric. Having stated that, an intent to measure the quality of test cases is described in [5].

Two different test case development approaches are *requirement-driven test case development* and *test-driven test case development*. In more traditional development testing, including the writing of test cases, requirements specifications are finished and the project is code complete before testing begins. More recent software development methods require testing and test cases to be defined and executed as the developers complete each part of the application. Which of the two approaches that provides the best test cases depend on who you ask and in what domain that person works. An experiment aimed at measuring and comparing the quality of test cases produced in a requirement-driven test case development environment against the quality of test cases produced in a test-driven test case development environment can be found in [3]. No conclusive result concerning any statistically proven difference in quality was presented, though.

[1]Test subject could range from a piece of code (in unit testing) to a general functionality (in system testing).

Software products encountered in safety- or security-critical domains, such as avionics, automotive, train, nuclear, medicine, banking, etc. are mostly requirement-driven, meaning that a set of requirements, sometimes depending on one or several international standards, impact on the software development and, indirectly, also impact on the test cases developed. Other types of application domains are less critical, such as GUI's, cellular phone apps, etc., thus leaving a great deal more of individual freedom to the software and test designers. For obvious reasons, these two different approaches, when it come to writing test cases, also impact the way test case designers think about what defines a good test case. The study presented in continuation have had a focus on the former approach of preparing test cases.

3 THE STUDY

Based in part on the work by Kaner [4] and Atmadja et al. [2], 15 criteria, that from the authors perspective define "good" test cases, were formulated and then evaluated and ranked by a group of Swedish software experts. The range that was used to measure the conformity among the experts with regard to the presented criteria, went from 0 to 10, where 0 indicates "not at all relevant" and 10 indicates "extremely relevant". The numbers within parentheses in the following list correspond to the numbering of the criteria in Sect. 2.

- Accurate: the outcome of a test case should be predictable and acquirable (15)
- Efficient: a test case should not require too many resources regarding staff or hardware/software (5)
- Independent: a test case should be able to execute without depending on other test cases for its success (0)
- Maintainable: a test case should be relatively easy and inexpensive to handle (0)
- Repeatable: a test case should produce the same result if run multiple times (16)
- Reusable: a test case could be reused, if necessary (0)
- Correct: a test case should not include any ambiguity in its execution (13)
- Clear: a new tester should not require any help to execute a test case (5)
- Complete: a set of test cases for a specific SW product should ensure 100% requirement coverage (6)
- Covering: a set of test cases for a specific SW product should ensure 100% code coverage (0)
- Consistent: similar test cases should be written in a consistent fashion (0)
- Simple: a test case should not include unnecessary steps or words (14)
- Powerful: a good test case is more likely to discover bugs than a not so good test case (1,2)
- Traceable: a test case could be traced to its corresponding requirement (6)
- Compact: a test case should cover only 1-2 requirements (14)

A (0) indicates a "new" criteria. As can be observed in the list, there are five "new" criteria, not covered by Kaner or Atmadja et al., i.e. *Independent, Maintainable, Reusable, Covering*, and *Consistent*. All these criteria are highly relevant, not to be taken for granted,

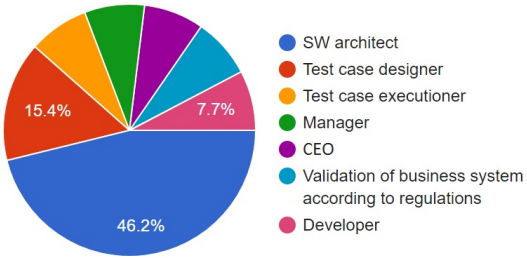

Figure 1: Relation to SW development and testing

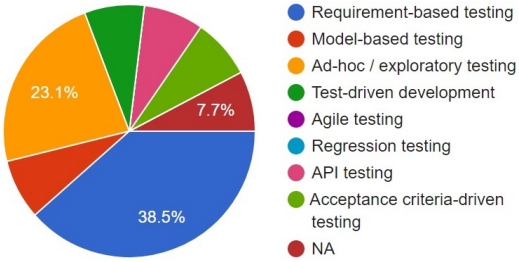

Figure 2: Type of testing performed

and were therefore made part of the study. Numbers (3), (4), (7), (8), (9), (10), (11) and (12) are not in the list as they, accordingly to the authors, belong to a completely different organizational level altogether and should not be expected to be handled and/or solved during testing. For example, *Help managers make ship/no-ship decisions*, should be handled by management, not by testers. It is a topic that is too important and too complex to be solved by a tester. Another example is *Assure quality*; quality is something that has to be build into the development process and cannot be left until the end, during the testing phase. However, it could be considered a feasible criteria if a company would apply test-driven test case development, where testing comes first, followed by the development of the object code. Finally, the criterion *Powerful* has joined two other criteria, *Find defects* and *Maximize bug count*, as they both relate to the same thing, bugs.

Next step in the study was to have the criteria evaluated and ranked by Swedish software experts (working with testing in their daily activities) using a questionnaire. 53.8% of the participants in the study had a previous experience of 5 to 10 years while 30.8% had more than 10 years of experience. The distribution of the experts background is shown in figure 1. The experts relation to software development and testing is presented in figure 2. The results from the questionnaire are presented in table 1.

As a part of the questionnaire, the experts were invited to comment on the criteria. Some of their remarks are shown below:

- Accurate: *It's rare that the the outcome is predictable, it varies on many factors as quality of requirements, maturity of SW, satellite systems and integrations.*
- Efficient: *Since time is a rare resource and availability of HW and SW for execution should be as easy to accommodate as possible, efficiency is of essence.*

Table 1: Ranking of good test case criteria

Rank	Criteria	Mean	Std dev
1	Repeatable	9.2	1.0
2	Accurate	8.4	1.5
3	Correct	8.1	1.5
4	Powerful	7.5	2.1
5	Maintainable	7.4	2.5
6	Complete	7.2	1.7
7	Traceable	7.2	2.3
8	Consistent	6.8	2.4
9	Reusable	6.5	2.2
10	Simple	6.3	2.2
11	Efficient	6.3	2.2
12	Clear	6.2	2.7
13	Independent	5.8	2.5
14	Covering	5.6	2.1
15	Compact	4.5	2.3

- Independent: *With independent test cases you are more likely to be able to execute them in a feasible way. They are also easier to use as building blocks to build flows.*
- Maintainable: *Maintenance is costly and test cases should be as easy as possible to maintain. Ideally, you can change a few parameters when necessary and not going through too many lines.*
- Repeatable: *With the same input it should be repeatable, but a test case outcome might vary depending on if the input is exactly the same or slightly different. If used in regression testing, you do want the same result from the same input time after time after time.*
- Reusable: *Time is a rare resource which makes reusable test cases very attractive.*
- Correct: *If the goal is to test a strict set of instructions you would like it to be very clear. Sometimes you want it to be a bit vague so the executioner might take different ways to a specific goal and make decisions of themselves.*
- Clear: *A clear test case is very valuable since the cost to introduce the executioner to the test case can be kept fairly low*
- *Complete: The goal is, of course, to reach 100% requirement coverage. However, sometimes you need to focus on the areas of largest risk.*
- Covering: *Depending on how well requirements and code are documented, it might be very hard to get 100% code coverage. Often, this is made by the developers by using specific tools, and even then it is rare to get more than 80% code coverage.*
- Consistent: *If the test cases are consistent they are much easier to combine and the cost of switching between test cases (context switch) becomes much lower.*
- Simple: *What is simple varies a lot so deciding what is a unnecessary step can be very hard. Sometimes you need several small steps to make it easy to understand and to catch the right result, sometimes fewer and simplified steps is what is most feasible.*

- Powerful: *It really depends on the goal of the test case, is it to proof that happy path works, or is to find severe bugs. It's rather rare that you can combine those two.*
- Traceable: *In the cases where you have clear requirements it is of great value if it is traceable to the requirements. In those cases where you have vague or non requirements or similar documentation you can't reach traceability to the requirements, but in those cases it is important that you have documented the test so it is traceable regarding to what steps you have made.*
- Compact: *If time and resources allow you, it is good to have compact test cases that are possible to use as building blocks.*

At the end questionnaire, the participants were invited to write free-text comments related to testing in general and "good" test case criteria in specific. Below can be found an excerpt consisting of three of these open comments.

> *'The test case should be based on a requirement. The test case should be unambiguously repetitive and if executed again, the result should be the same each time. It should be measurable and no uncertainty should exist as to the reason for executing the test case or its purpose.'* (SAAB, aerospace industry)

> *'We want to observe a clear connection to requirements. The difficulty lies in obtaining a clear image of the demands on all levels (component, system, production). Manually executed test cases should be simple and clear. Automatically executed test cases should not be too complex or cover too many requirements (there exist a risk that you have similar behaviors for different input data, something that generates many IF-statements.'*
> (Husqvarna, outdoor power tools)

> *'I have worked 30 years with software in embedded systems, of which 20 as a consultant. The test cases that provide the best results, i.e. detects most errors, are those that test combinations of input data with extreme or out-of-range values. That is, test cases that go beyond the most obvious. Unfortunately, specifications are usually written based on requirements focusing on what should happen with valid input data in simple use-cases. This is also what programmers most often focus on, the typical case being unit tests. Those tests often provide nothing or very little. On the contrary, if you manage to create test cases that involve "strange" use-cases, they often detect errors right away and often put the light on flaws in the software design.'*
> (Atollic, embedded systems)

The first two comments are in line with what would usually be expected, that test cases should have a clear connection with the requirements. What sticks out is the third answer that puts the focus more on exploratory testing. Requirements-driven testing does not exclude test-driven (or exploratory) testing but still, it is an interesting comment coming from an embedded systems company.

4 CONCLUSIONS

Most of the criteria not covered by Kaner or Atmadja et al. (marked with a (0) in the previous list), received a fairly high ranking by the experts, as shown in table1. The comments by the experts also support this statement. For example, *Maintainable*, received a mean value of 7.4 (std. dev. 2.5) and one expert pointed out that *only having to change a few parameters when necessary and not having to go through too many lines is something to strive for.* Another example, *Clear*, received a mean value of 6.2 (std. dev. 2.7) and one comment was that *a clear test case would lower the threshold between the test designer and the test executioner.*

Three of the criteria that is normally associated with requirement-driven software development and testing, i.e. *Complete, Covering* and *Traceable*, end up behind other types of criteria. This could be attributed to the fact that many software developers, both of stand-alone software products and embedded systems, want and need more answers from their tests than just a simple "test passed". The requirement-driven tests remain but the test-driven tests move in beside the former type of tests. The top criteria, *Repeatable*, receives 9.2 on the ranking scale with a standard deviation of only 1.0. Possibly a surprise to some but maybe not in the light of the trend, moving towards test-driven tests where a need for repeatability is of utmost importance.

Regarding the more "managerial" criteria that were not included in this study, a diligent reader might argue that as managers and CEOs answered the questionnaire, as indicated in figure 2, they would be biased and put lower grades for the not so "managerial" criteria. This did not occur, though, as their individual grading did not indicate any tendency towards that. Furthermore, the questionnaire also included a free-text comment field and no participant in the study seemed to lack any missing "managerial" criteria.

ACKNOWLEDGMENTS

The work presented in this paper was funded by the Knowledge Foundation in Sweden, grant KKS-20140170.

REFERENCES

[1] IEEE Standards Association et al. 2010. Systems and software engineeringâĂŤVocabulary ISO/IEC/IEEE 24765: 2010. *Iso/Iec/Ieee* 24765 (2010), 1–418.
[2] Michael Atmadja and Richard Zhang Shuai. 2011. Chapter 3: Testing. In *A Fresh Graduate's Guide to Software Development Tools and Technologies*, Damith C. Rajapakse (Ed.). School of Computing, National University of Singapore, 1–21. http://www.comp.nus.edu.sg/~seer/book/2e/
[3] Adnan Čaušević, Daniel Sundmark, and Sasikumar Punnekkat. 2012. Test case quality in test driven development: a study design and a pilot experiment. In *Proceedings of the EASE 2012*. IET, 223–227. https://doi.org/10.1049/ic.2012.0029
[4] Cem Kaner. 2003. What is a good test case?. In *Software Testing Analysis & Review Conference (STAR) East, Orlando, FL*. 1–16. http://kaner.com/pdfs/GoodTest.pdf.
[5] Christian Pfaller, Stefan Wagner, Jörg Gericke, and Matthias Wiemann. 2008. Multi-dimensional measures for test case quality. In *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on*. IEEE, 364–368.
[6] He Tan, Anders Adlemo, Vladimir Tarasov, and Mats Johansson. 2017. Evaluation of an application ontology. In *Proceedings of the Joint Ontology Workshops 2017 Episode 3: The Tyrolean Autumn of Ontology, JOWO 2017*. CEUR-WS.org, 14. http://ceur-ws.org/Vol-2050/DEW_paper_1.pdf
[7] He Tan, Ismail Muhammad, Vladimir Tarasov, Anders Adlemo, and Mats Johansson. 2016. Development and evaluation of a software requirements ontology. In *Proceedings of 7th International Workshop on Software Knowledge-SKY 2016, in conjunction with the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management-IC3K 2016*. SCITEPRESS, 11–18.
[8] Vladimir Tarasov, He Tan, Ismail Muhammad, Anders Adlemo, and Mats Johansson. 2017. Application of inference rules to a software requirements ontology to generate software test cases. In *OWL: Experiences and Directions–Reasoner Evaluation: 13th International Workshop, OWLED 2016, and 5th International Workshop, ORE 2016, Bologna, Italy, November 20, 2016, Revised Selected Papers*, Vol. 10161. Springer, 82.
[9] Erik van Veenendaal and Brian Wells. 2012. *Test Maturity Model Integration TMMi*. Uitgeverij Tutein Nolthenius, The Netherlands.
[10] Ole Zimmermann. 2017. *Modelling complex software requirements with ontologies*. Master's thesis. Universität Rostock. Part of the OSTAG project.