Fault-Detection Sensitivity Based Assessment of Test Sets for Safety-Relevant Software

Susanne Kandl
Institute of Computer Engineering
Vienna University of Technology
Austria
Email: susanne@vmars.tuwien.ac.at

Jean-Marc Forey
Synopsys, Inc.
Grenoble
France
Email: Jean-Marc.Forey@synopsys.com

***Abstract*—In testing it is, in general, not possible (or at least extremely time-consuming) to cover the complete input data space, therefore usually a test set is selected for a given coverage criterion (like decision coverage or similar). This restricted test set covers only a part of the complete input data space with a degraded fault-detection capability. The fault-detection capability of a test set is given by the number of *detected* faults in relation to the number of *actual* faults. In this work, we present a novel approach for the assessment of test sets based on their fault-detection sensitivity. The main goal of an efficient testing process is to reduce the test effort while targeting a maximum number of detected faults, i.e., an ideal test set requires a minimal execution time for the test run (defined by the number of the test cases and their individual run-times) with a maximum fault-detection sensitivity. Therefore, our proposed test-set selection process is not guided by a coverage criterion, but by the fault-detection capability of the different test cases using the output of the tool *Certitude Functional Qualification System*. We will apply our strategy on a safety-relevant (regarding ISO 26262) case study from the automotive domain focusing on the scalability of the test-set selection strategy. Our work presents a novel method to optimize the testing effort for software used in dependable systems with respect to a decreased testing effort while sustaining a high fault-detection capability.**

Keywords–Dependable Systems, Testing, Fault-Detection Sensitivity, Safety-Relevant Software.

I. INTRODUCTION

The Verification and Validation (V&V) of safety-relevant systems (a class of dependable systems) requires a tremendously high amount of effort [1]. Besides techniques like analysis or review, testing is one of the main methods to prove the correctness of the system. The project *Verification and Testing to Support Functional Safety Standards* (VeTeSS) [2] deals with the development of standardized tools and methods for the verification of safety-relevant systems in the context of ISO 26262 (Functional Safety for Road Vehicles) [3]. One main aspect is to tackle the contradiction between *reducing the V&V-effort* while *enhancing the dependability* of the safety-relevant components. Usually, there is a correlation between the testing effort and the confidence in the test process (the test result, respectively) illustrated in the exemplary Figure 1. An increasing testing effort results in a higher confidence in the test process. After passing a critical mark in the testing effort (indicated by the dashed line), the confidence in the testing result increases only marginally approaching 100% confidence in an asymptotic way for additional testing effort.

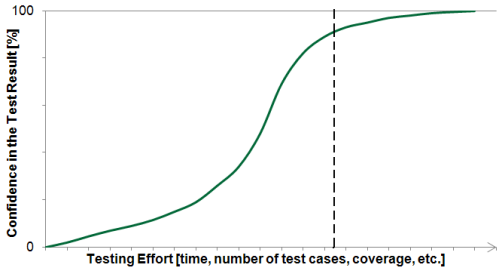

Figure 1. Correlation Between the Testing Effort and the Confidence in the Test Process

The selection of the test set (the set of test cases) has a significant impact on the testing effort and the resulting test result. The example given in Figure 2 demonstrates this fact (TD means Test Data: some input data to execute this branch; TC stands for Test Case: an execution trace in the program). The control flow of this example consists of 4 if-decisions. Each decision has 2 branches, i.e., in the overall we have to cover 8 branches for 100% Decision Coverage (DC). For that we need test cases to cover all the branches (b_1 + b_2, b_3 + b_4, b_5 + b_6, b_7 + b_8). Due to the redundancies in the traces this results in 5 test cases (TC_1, TC_2, TC_3, TC_4, TC_5). Starting with Test Set TS_{INI} consisting of TC_1 and TC_5 shows that 4 out of 8 branches are covered, that means we achieve 50% DC. Adding the test case TC_2 to TS_{INI} results in covering 5 of the 8 branches, this equals 62,5% DC (i.e., an increase of 12,5%). Adding the test case TC_3 to TS_{INI} results in covering 6 of the 8 branches, this equals 75,0% DC (i.e., an overall increase of 25,0%). That means that both extended test sets contain 3 test cases, but the latter one achieves better results regarding the coverage criterion DC. Although TC_3 is longer than TC_2 (incorporating more test steps), usually the length of the test cases has a negligible effect on the testing effort, as the initialization of the test cases is the most time-consuming part in the test-case execution.

The assessment of a test set based on a given coverage criterion is a very common technique but studies (like [4] [5] [6]) indicate that structural coverage is a rather insufficient means to determine the quality of the test set. Even sophisticated coverage criteria like MC/DC (Modified Condition/Decision Coverage) do not guarantee a high error-detection sensitivity (see [7] [8]). What we are *really* interested in is the *fault-detection sensitivity* of a test set, that means the number of *detected* faults in relation to the number of *actual* faults for

 ISBN: 978-1-61208-378-0

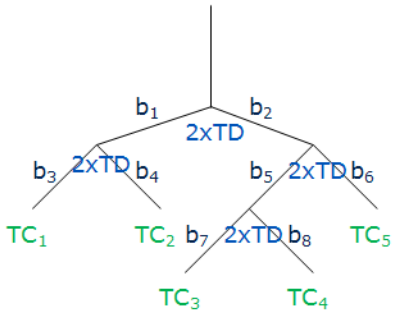

Figure 2. Example for Test-Set Selection for Decision Coverage

a given test set. Faults are erroneous parts in the program (colloquially aka: bugs), for instance, a mutated name of a variable (`variable_1` instead of `variable_2`). This motivated the idea of a fault-detection sensitivity based assessment of test sets. The main principle is to select the test set on the basis of the powerfulness of test cases to detect faults. For this we are using the analysis of the Certitude Functional Qualification System [9]. This tool introduces artificial faults into a program and determines which faults are covered by specific test cases. Some of the test cases are capable to reveal a high number of faults, whereas other test cases detect only a few faults. The target is to assess a test set regarding the fault-detection sensitivity (given by the capability of the test cases to detect faults). To evaluate the feasibility of this idea for a real industrial case study we apply our strategy to a case study from the automotive domain (software for the selection of the driving mode for an electric car).

In the remaining part of the paper, we give some basic definitions (Section II) and describe the functionality of the tool Certitude to explain how we are using this tool for our approach (Section III). In the following, we elaborate our strategy for fault-detection sensitivity based assessment of test sets and present the questions addressed by our study (Section IV). Finally, we give a short overview on our work in progress to evaluate our approach on a case study (Section V) and conclude with a summary of the paper and planned future work (Section VI).

II. Basic Definitions

A **Test Set** TS consists of *test cases*. A **Test Case** TC is a *trace* in the execution of the program defined by the variable values *including* the correct output for given input data. Example: A test case for a function `sum(a, b)` may be $(2; 3; 5)$. The **size** of a test set $S(TS)$ is defined by the number of test cases.

A **fault** is a mutation of the program (i.e., a deviation of the correct implementation). We will consider faults for operators, variables, and values.

A **strong** test case is a test case that is capable to detect many faults. A **weak** test case is a test case that is capable to detect only a few faults. The fault-detection sensitivity of a test case $F(TC_x)$ is defined by the number of faults that are detected by this test case. The fault-detection sensitivity of a test set $F(TS)$ is defined by the sum of the detected faults of the test cases in TS, i.e., $F(TS) = \sum_{x=1}^{S(TS)} F(TC_x)$.

The classification of test cases into *strong* and *weak* test cases depends on the achieved values for $F(TC_x)$ for the test cases in TS: A test case with $F(TC_x) = 10$ may be a *strong* test case in a test set with test cases with a maximum fault-detection sensitivity of 12, but it is rather a *weak* test case when other test cases in this test set are able to detect, e.g., 100 faults.

In the following, we will also distinguish between an **Easy-To-Detect (ETD) fault** and a **Not-Easy-To-Detect (NETD) fault**. Whereas the first one is detected by many test cases (for instance, 27 out of 100 test cases), the second one is only detected by a few test cases (for instance, 2 out of 100 test cases). We define the number of test cases of a test set that are capable to detect a specific fault (Ftl_y) $D(Ftl_y) = \#TC_x$ (for $x = 1...S(TS)$) with TC_x detects the fault Ftl_y, as the metric to guide the classification of faults. The precise distinction between *EDT* and *NEDT* faults depends on the concrete values for a case study (the program under test and the applied test set).

III. Certitude Functional Qualification System

Basically, the tool Certitude Functional Qualification System by Synopsys Inc. [9] intends to measure the effectiveness of a verification environment. It is able to identify verification weaknesses that allow bugs to stay undetected in the testing process and may lead to functional problems. The basic principle of Certitude is to introduce mutations (i.e., artificial software faults) into the design and prove whether the verification environment (the test set) is capable to detect these faults or not, see Figure 3 (RTL refers to Register-Transfer Level). Please, consider that this technique is different to typical software fault injections with the intention to validate fault-handling mechanism at runtime and to evaluate the way a system behaves in the presence of faults (like [10]). The aim of Certitude is to determine the capability of a verification environment to detect design mutations (similar to [11] and [12]).

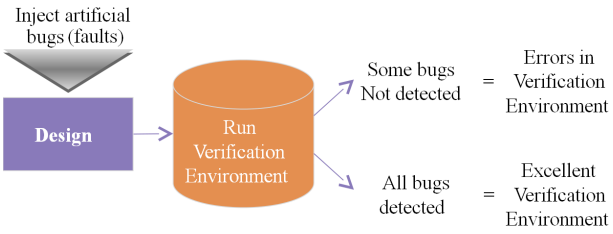

Figure 3. Certitude Functional Qualification Process

If all (or at least most) of the artificially introduced bugs are identified by the verification environment, this V&V-environment is proven to be *good*. If many of the artificially introduced bugs are *not* identified by the verification environment, this indicates that the V&V-environment is insufficient. Either some test cases (or specific test scenarios) are missing or wrongly implemented, or the checks are not robust enough (missing or broken). Certitude provides two different modes: 1. The *verification improvement mode* analyzes the verification of the design and identifies specific holes and weaknesses. 2. The *metric mode* allows to measure the overall quality of the verification environment in a quantitative way using a statistical approach.

 ISBN: 978-1-61208-378-0

Certitude combines static analysis with mutation-based techniques for introducing mutations (i.e., artificial bugs) into the system under test, see Listing 1 for an example. In line 1, the original version is given. Then the Boolean operator *OR* (|) is mutated to the Boolean operator *AND* (&) resulting in the faulty version in line 2.

```
1    a = b | c;
2    a = b & c;
```

Listing 1. Original Code and Faulty Program Code

After introducing the mutations, Certitude determines whether the V&V-environment can activate (i.e., exercise) the faulty code, propagate the effects to an observable point, and detect the presence of the fault. This is done in three phases (see Figure 4): a) In the fault model analysis phase, the design is analyzed and appropriate faults are selected that will be injected into the system. b) In the fault activation phase, the specified tests (selected from the regression) are run once and the behavior of the V&V-environment with respect to the faults is analyzed. c) In the fault detection phase, selected test cases from the V&V-environment are executed to measure the ability of the V&V-environment to detect the faults.

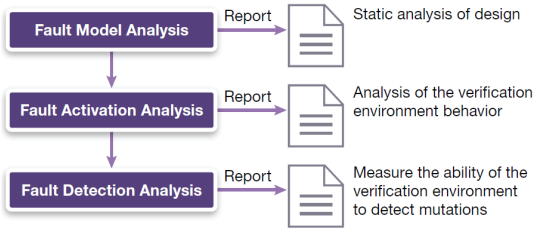

Figure 4. Phases of the Certitude Functional Qualification

Furthermore, Certitude uses a proprietary algorithm to automatically classify and prioritize the faults (the injected faults are qualified in a priority order). The subsequent qualifications contain the results from the previous test runs and focus on the remaining undetected faults. By this, it is possible to find and fix weaknesses in the V&V-environment in an early stage of the verification process, expand the set of qualified faults (as both, the V&V-environment and the design, evolve), achieve an incremental improvement over time, and minimize the effort for analysis and debug. *Latent* faults are mutations that have no impact on the program behavior. Some of them can be recognized by Certitude in the diagnosis of the results of the test run. At the moment, Certitude supports hardware description languages, like VHDL [13] and Verilog [14], and programming languages like C/C++.

Within the analyses of Certitude the tool provides a list that shows for each executed test case the number of *activated*, *propagated*, and *detected* faults. The code skeleton given in Listing 2 demonstrates the meaning of the three different terms: In line 2, the correct program version is given (with `i<11`) and in line 3, the mutated (faulty) program version is given (with `i<=11`). For the activation of this fault we need a test case with the variable `a` equals the value *TRUE*, otherwise the fault is not activated at all. The propagation of a fault to an observable failure is necessary to observe (and thus, detect) the actual fault. A fault may cause an error (an invalid state in the system behavior). An error may cause further errors (therefore an error may act as a fault), or it may propagate and then be observable. To propagate the injected fault, we need a test case with the value 5 for the variable `j`. Only then we are able to observe a deviation from the original program behavior caused by the introduced fault.

```
1    while (a == TRUE) {
2       for(i=1; i<11; i++) {    //correct version
3       for(i=1; i<=11; i++) {   //mutation
4          print (i);
5          if (j==5) {
6             product = i * j;
7             print (product);
8          }
9       }
10      if (product > 50)
11         print (error_message);
12      else
13         print (OK);
14   }
```

Listing 2. Original Code and Faulty Program Code

The main output of Certitude for the quality assessment of the V&V-environment (the underlying test set, respectively) is the number of detected and undetected faults. If the ratio of detected faults to the overall number of injected faults is high, this indicates a matured and reliable test set, otherwise the test set has to be improved. For further analysis, Certitude also provides a list showing the number of activated (*#ActF*), propagated (*#PropF*), and detected faults (*#DetF*) for each test case, see the exemplary Table I.

TABLE I
EXAMPLE OUTPUT OF CERTITUDE

Test Case	*#ActF*	*#PropF*	*#DetF*
TC_1	80	70	50
TC_2	60	50	20
TC_3	20	10	5
etc.	etc.	etc.	etc.

Assuming a total number of introduced faults of 100, TC_1 appears to be a *strong* test case, whereas TC_3 is a rather *weak* test case regarding the lower number of detected faults.

The main purpose of the Certitude Functional Qualification System is to provide the information that allows the test engineer to improve the overall V&V-environment (i.e., the test set) by identifying weaknesses of the V&V-environment (e.g., missing test cases) and improving the V&V-environment (by adding these missing test cases). In the following, we present an idea to use the tool Certitude for the selection of test sets that require less testing effort while sustaining a high fault-detection sensitivity.

IV. FAULT-DETECTION SENSITIVITY BASED ASSESSMENT OF TEST SETS

The main idea of our strategy for the selection of a test set is to include the test cases with a high fault-detection sensitivity $F(TC_x)$ and to dismiss the test cases with a low $F(TC_x)$. By this, we achieve a smaller test set (thus, reducing

 ISBN: 978-1-61208-378-0

the overall test effort). The fault-detection sensitivity based assessment of test sets is instrumented by a) a parameter for the number of detected faults by the definition of a threshold-value T_{DetF} for detected faults: if $F(TC_x)$ is greater than T_{DetF}, then TC_x is included in the final test set otherwise not. or b) a parameter for the targeted reduction of the test set by defining a concrete value for the reduction (e.g., 20% of the original test cases are omitted).

In this study, we are mainly interested in:

- What is the exact effect on the degradation of the fault-detection sensitivity of the new test set $F(TS)$ for a) and b)? It may happen that a test case with a low $F(TC_x)$ is discarded without any (negative) impact on $F(TS)$ because the related faults are anyway covered by other (remaining) strong test cases. On the other hand omitting a weak test case may have a significant impact on $F(TS)$ because this test case is required for the activation of a couple of faults (without activation of these faults they cannot be detected by the strong test cases).
- What is the relation between the achieved reduction regarding the test effort (correlating with the size of the test set) and the resulting degradation of the fault-detection sensitivity of the test set $F(TS)$? If we risk to fail to detect important faults just by omitting 5 out of 1000 test cases (i.e., the effort reduction is almost negligible), then the efficiency of the strategy is questionable. Preliminary empirical results indicate that omitting some of the weak test cases results in a significant reduction of the test effort while degrading the actual fault-detection sensitivity $F(TS)$ only marginally.
- Experiences so far suggest that for our analysis we have to differ between *Easy-To-Detect (ETD)* faults and a *Not-Easy-To-Detect (NETD)* faults. What does a low/high number of ETD-/NETD-faults means for our strategy? For a system with many ETD-faults, in general, the omission of a weak test case has only a tiny effect on the overall $F(TS)$. This is not valid for NETD-faults. Certitude also provides implicitly some information to distinguish between ETD- and NETD-faults. This information can support the evaluation of our strategy.
- Is our strategy (due to computation time) feasible for a real industrial case study? In general, determining a minimal test set is decidable but, the problem is NP-complete (that means the algorithm requires an exponentially high effort), thus usually heuristics are applied to find an optimal test set [15]. We try to avoid this problem by only a few test cycles to determine $F(TS)$ and the values for $F(TC_x)$ for the initial test set. Selecting the test cases for the improved test set is linear and then we are executing the test runs for the reduced test set.

V. Evaluation on an Industrial Use Case

In our empirical evaluation, we focus on automotive software written in the programming language C. One of our industry partners provides a use case for an automatic gear selection for an electric car. The Electric Vehicle Controller (EVC) manages the functions for the gear selection of the available states (for instance: Park; Reverse; Neutral; and Drive, the function similar to automated transmission). The main function of the use case allows the selection of the driving mode from the four different available states. The initial test set is generated automatically from a SysML-model of the use case. With the final results of the empirical evaluation of our strategy for fault-detection sensitivity based assessment of test sets, we will be able to give clear recommendations for the applicability, the benefits, and also the limitations of our novel idea.

VI. Conclusion and Future Work

Finding the optimal test set (minimal size and maximum fault-detection capability) is a permanent ongoing challenging task. Experienced test engineers may be able to write down a few good test cases (good in finding faults), some of them especially tuned to detect the faults/errors of a certain program developer or adapted to intricate (thus error-prone) parts of the system under development. Coverage metrics are often used to determine the maturity of the test set based on the assumption that an increase in the coverage induces a better fault-detection sensitivity. In this work we present a novel idea for the assessment of the quality of a test set by the main attribute we are interested in, namely the *fault-detection sensitivity*. For our strategy, we made use of the tool Certitude Functional Qualification System that provides us with the necessary information about the fault-detection capability of the different test cases of a test set. Static analysis combined with mutation-based fault injection yields in precise information about the powerfulness/weakness of a test case to detect faults. This information can be used for a qualified selection of test cases to reduce the size of the test set while preserving a desired test set quality (regarding the fault-detection sensitivity). Our idea intends also to motivate a test process stronger guided by fault injection instead of using fault injection only as a recommended method prescribed by standards for safety-relevant systems, like ISO 26262.

Besides finishing the analysis for the mentioned use case, we plan to extend the evaluation of our strategy to (at least) another case study. Not only the number, but also the type of faults and the program structure influence significantly the fault-detection sensitivity of a test set, so precise answers to the listed questions can only be given after results from different use cases. We also consider to address faults leading to a multiple point failure (individual faults may lead only in combination with another independent fault/with other independent faults to a failure, the so-called multiple point failure). This kind of faults are usually difficult to detect (as they are only observable in the presence of other faults).

Acknowledgments

This work has been partially funded by the ARTEMIS Joint Undertaking and the National Funding Agency of Austria for the project VeTeSS under the funding ID ARTEMIS-2011-1-295311.

 ISBN: 978-1-61208-378-0

REFERENCES

[1] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, 2nd ed., ser. Series: Real-Time Systems Series. Springer, 2011.

[2] "ARTEMIS VeTeSS: Verification and Testing to support functional Safety Standards," 2014, last visited: 09-22-2014. [Online]. Available: http://www.vetess.eu

[3] ISO: International Organization for Standardization, "ISO 26262: Functional safety – road vehicles," 2011.

[4] M. Staats, G. Gay, M. Whalen, and M. Heimdahl, "On the danger of coverage directed test case generation," in Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering, ser. FASE'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 409–424, last visited: 09-22-2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28872-2_28

[5] K. Kapoor and J. Bowen, "Experimental evaluation of the variation in effectiveness for DC, FPC and MC/DC test criteria," Proceedings International Symposium on Empirical Software Engineering, ISESE 2003, Sept.-1 Oct. 2003, pp. 185–194.

[6] J. Guan, J. Offutt, and P. Ammann, "An industrial case study of structural testing applied to safety-critical embedded software," in Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ser. ISESE '06. New York, NY, USA: ACM, 2006, pp. 272–277, last visited: 09-22-2014. [Online]. Available: http://doi.acm.org/10.1145/1159733.1159774

[7] A. Dupuy and N. Leveson, "An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software," in Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th, vol. 1, 2000, pp. 1B6/1–1B6/7 vol.1.

[8] S. Kandl and R. Kirner, "Error detection rate of MC/DC for a case study from the automotive domain," LNCS 6399: S.L.Min et al.(Eds.): Proceedings of the 8th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2010), Oct. 2010, pp. 131–142.

[9] Synopsys Inc., "Certitude - functional qualification system," 2014, last visited: 09-22-2014. [Online]. Available:https://www.synopsys.com/TOOLS/VERIFICATION/FUNCTIONALVERIFICATION/Pages/certitude-ds.aspx

[10] D. Alexandrescu, L. Sterpone, and C. Lopez-Ongil, "Fault injection and fault tolerance methodologies for assessing device robustness and mitigating against ionizing radiation," in Test Symposium (ETS), 2014 19th IEEE European, May 2014, pp. 1–6.

[11] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," IEEE Trans. Softw. Eng., vol. 32, no. 9, Sep. 2006, pp. 733–752, last visited: 09-22-2014. [Online]. Available: http://dx.doi.org/10.1109/TSE.2006.92

[12] J. Duraes and H. Madeira, "Emulation of software faults: A field data study and a practical approach," Software Engineering, IEEE Transactions on, vol. 32, no. 11, Nov 2006, pp. 849–867.

[13] P. J. Ashenden, The Designer's Guide to VHDL, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[14] S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2003.

[15] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," ACM Trans. Softw. Eng. Methodol., vol. 2, no. 3, Jul. 1993, pp. 270–285, last visited: 09-22-2014. [Online]. Available: http://doi.acm.org/10.1145/152388.152391

 ISBN: 978-1-61208-378-0