An Industrial Case Study on Fault Detection Effectiveness of Combinatorial Robustness Testing

Konrad Fögen, Horst Lichter

Research Group Software Construction, RWTH Aachen University, Aachen, Germany
https://www.swc.rwth-aachen.de

Abstract

Combinatorial robustness testing (CRT) is an extension of combinatorial testing (CT) to separate test suites with valid and strong invalid test inputs. Until now, only one controlled experiment using artificial test scenarios was conducted to compare CRT with CT. The results indicate advantages of CRT when much exception handling is involved. But, it is unclear if these advantages are also valid in the real-world. In this paper, we present the results of a case study conducted to compare the fault detection effectiveness of CRT and CT by testing an industrial system with 31 validation rules and 13 injected faults.

Keywords

Software Testing, Combinatorial Testing, Robustness Testing

1. Introduction

Robustness is an important property of software. It describes “the degree to which a system [...] can function correctly in the presence of [invalid inputs]” [1]. Invalid inputs are caused by external faults, i.e. faults in other systems or made by users interacting with a system. Examples are inputs to the system under test (SUT) that contain invalid values like a string value when a numerical value is expected, or invalid value combinations like a begin date which is after the end date. When invalid inputs remain undetected, they can propagate to failures in the SUT resulting in abnormal behavior or crashes [2].

Developers attempt to improve robustness of systems by implementing exception handling (EH) to detect and recover from invalid inputs. Unfortunately, EH is itself a significant source of faults (cf. [3, 4]). Therefore, it is important to test the exceptional behavior as well.

Combinatorial testing (CT) is a black-box test method that is based on an input parameter model (IPM) [5]. When considering the exceptional behavior, an IPM must describe invalid values and invalid value combinations that trigger EH. Unfortunately, invalid values and invalid value combinations can cause *input masking* (cf. [6, 7, 8]). When a SUT is stimulated with an invalid input, the EH is expected to detect it, to respond with an error message, and to terminate the SUT without resuming the normal behavior. Consequently, the remaining values and value combinations of the test input remain untested as they are masked.

To avoid input masking, combinatorial robustness testing (CRT) is developed as an extension to CT using a robustness input parameter model (RIPM) being an extension of an IPM with additional semantic information to annotate values and value combinations as invalid [7]. With this semantic information, valid test inputs can be selected which do not cover any invalid value or invalid value combination. Further on, strong invalid test inputs can be selected which contain exactly one invalid value or one invalid value combination.

Due to the separation of valid and strong invalid test inputs, the input masking effect can be avoided when testing the normal behavior and the exceptional behavior. However, in comparison to CT which does not separate valid and strong invalid test inputs, CRT requires effort to model the additional semantic information.

Despite the presence of input masking, CT can still be effective in detecting faults as a previous controlled experiment indicates [8]. Nevertheless, the fault detection effectiveness (FDE) of CT decreases for systems with much EH. Even for high testing strengths and large test suites, the FDE of CT deteriorates. For systems with much EH, CRT is a promising approach that can achieve a higher FDE while requiring fewer test inputs than CT [7]. For systems with little EH, CRT is at least as effective as CT.

Although, the current assessment is solely based on one controlled experiment with artificial test scenarios (cf. [7]). Therefore, our objective is to further compare CRT with CT guided by the following two research questions.

RQ 1 Is the CRT test method applicable in real-world test scenarios?

RQ 2 How does the CRT test method compare with CT in real-world test scenarios?

QuASoQ 2020: 8th International Workshop on Quantitative Approaches to Software Quality, December 01, 2020, Singapore

foegen@swc.rwth-aachen.de (K. Fögen);
lichter@swc.rwth-aachen.de (H. Lichter)
0000-0002-3440-1238 (H. Lichter)

To answer these research questions, we conducted a case study. According to Kitchenham et al. [9], a case study helps to evaluate the benefits of methods and tools in industrial settings. When applied to compare methods and tools, a case study is of explanatory nature "seeking an explanation of a situation or a problem" [10]. As Runeson & Höst state, a case study "will never provide conclusions with statistical significance" [10]. But it can provide sufficient information to help you judge if specific technologies will benefit your own organization or project" [9]. Since a case study has by definition a higher degree of realism than a controlled experiment [10], a case study that compares CRT with CT can provide additional insights that complement and extend the findings of the previously conducted controlled experiment.

The paper is structured as follows. Section 2 introduces basic concepts of CT and CRT. Related work is discussed in Section 3. Next, the design of the case study is introduced (Section 4) and its results are presented (Section 5). Afterwards, threats to validity are discussed (Section 6) before the paper is concluded in Section 7.

2. Background

In the following, CT and CRT are briefly introduced. For more information, please refer to [11, 5, 7].

2.1. Combinatorial Testing

CT is a black-box test method [5]. It is based on an **input parameter model** (IPM) which declares n parameters and each parameter is associated with a non-empty set of values. A **schema** is a set of parameter-value pairs for d distinct parameters [12]. A schema with $d = n$ parameter-value pairs is a **test input**. A schema a covers another schema b if and only if schema a includes all parameter-value pairs of schema b.

Real-world systems are often constrained and certain values should not be combined to schemata and test inputs [5]. These schemata are irrelevant because they are not of any interest for the test. Test inputs that cover irrelevant schemata are irrelevant as well and their test results have no informative value. Hence, they should be excluded from testing.

Constraint handling is often used to exclude irrelevant schemata [13]. Therefore, irrelevant schemata are explicitly modeled by a set of logical expressions (called **exclusion-constraints**). A schema is **relevant** if it satisfies all exclusion-constraints. A schema is **irrelevant** if at least one exclusion-constraint remains unsatisfied.

A **coverage criterion** is a condition that must be satisfied by a test suite. A **test selection strategy** describes how values are combined to test inputs such that a given coverage criterion is satisfied [11]. Test suites resulting from a test selection strategy that supports constraint handling, e.g. IPOG-C [13], satisfy the ***t*-wise relevant coverage criterion**. This criterion is satisfied if the relevant test inputs of a test suite cover all relevant schemata of degree $d = t$ that are described by an IPM [11, 5].

2.2. Combinatorial Robustness Testing

To avoid input masking, CRT is developed as an extension to CT that separates valid and invalid test inputs [7]. To better separate the concepts, we say that CT relies on IPMs while CRT relies on **robustness input parameter models** (RIPM). A RIPM contains additional **error-constraints** which is another set of constraints to annotate relevant schemata as invalid. A relevant schema is also a **valid schema** if it satisfies all error-constraints. A relevant schema is an **invalid schema** if at least one error-constraint remains unsatisfied. Further on, an invalid schema is a **strong invalid schema** if exactly one error-constraint remains unsatisfied.

Test selection strategies like ROBUSTA [7] not only consider exclusion-constraints to exclude irrelevant schemata. They also consider error-constraints and exclude invalid schemata from valid test inputs. Further on, strong invalid test inputs are selected such that each invalid value and invalid value combination that is modeled by error-constraints appears in strong invalid test inputs.

Valid test inputs are selected to satisfy ***t*-wise valid coverage**. The t-wise valid coverage criterion is an extension of the t-wise relevant coverage criterion. It is satisfied if all valid schemata with a degree of $d = t$ that are described by a RIPM are covered at least once by a valid test input.

Strong invalid test inputs are selected to satisfy ***b*-wise strong invalid coverage** where b denotes the robustness interaction degree. Without robustness interaction ($b = 0$), the coverage criterion is called single error coverage (cf. [11, 7]). It is satisfied if each invalid schema that is described by an error-constraint appears in a strong invalid test input. With robustness interaction ($b \geq 1$), each described invalid schema is combined with all valid schemata of degree $d = b$. The coverage criterion is satisfied if all combinations of invalid schemata and b-sized valid schemata are covered by strong invalid test inputs.

Following these brief introductions of CT and CRT, the conceptual difference between the two approaches should become clear. CT and CRT use the same parameters and values. But CT does not distinguish between valid and invalid schemata. Instead, both types of schemata are mixed and the FDE purely relies on the combinatorics, i.e. different testing strengths t. In contrast, CRT distinguishes valid and invalid schemata to avoid the effect of input masking. Here too the FDE relies on combinatorics but the avoidance of input masking has an additional influence.

CRT requires the effort to model error-constraints. Test selection strategies that consider error-constraints also become more complex. This raises the question whether the avoidance of input masking outweighs the additional effort and complexity of CRT. Until now, only artificial test scenarios are used to compare CT with CRT (cf. [7]) and it remains unclear if indicated advantages of CRT can be transferred to real-world scenarios. Therefore, this case study was conducted.

3. Related Work

To the best of our knowledge, Sherwood [6] first mentioned invalid values in the context of CATS which is a test selection strategy and tool for CT. Cohen et al. [14] and Czerwonka [15] also acknowledged the necessity to separate valid and strong invalid test inputs. They also published test selection strategies and tools and the IPMs contain semantic information to distinguish relevant from irrelevant schemata and to distinguish valid from invalid values. However, invalid value combinations are not directly supported. Therefore, we proposed ROBUSTA and the structure of RIPMs with error-constraints [7].

Many studies exist that demonstrate the usefulness and effectiveness of CT (cf. [16, 17, 18]). But most studies do not distinguish between relevance and validness and focus on testing the normal behavior.

One case study by Wojcıak & Tzoref-Brıll [19] reports on applying CT and also considers testing with invalid inputs. They report that single error coverage was not sufficient because EH depended on interactions between invalid and valid values. In particular, “the same [exception] would often be handled differently depending on the firmware in control [...] or depending on the configuration of the system”. A further remark is concerned with the ratio of valid versus invalid test inputs: “Since a lot of attention was given to [robustness] testing [...] where full recovery in the presence of [exceptions] was expected, the [test suite] contained a ratio of up to 2:1 [invalid test inputs vs. valid test inputs].”

4. Case Study Design

In this section, the case under analysis and the data collection procedure are introduced.

4.1. Case Under Analysis

The case is a development project conducted by an IT service provider of an insurance company, where a new software was developed to manage the life-cycle of life insurance contracts. One subsystem of the software is concerned with the validation of insurance application data according to a set of validation rules and with forwarding the data when it satisfies the validation rules. It is the same project which we analyzed in a previous case study (cf. [18]).

Altogether, 31 validation rules that are defined to check insurance application data. The order of the validation rules is predefined and all validation rules are traversed for each insurance application data. Whenever a validation rule is not satisfied by an insurance application, a corresponding error code is returned and the remaining validation rules are skipped. If all validation rules are satisfied, the subsystem returns SUCCESS and the insurance application data is further processed. Although, the further processing is out of scope for this case study.

Each validation rule is built as an implication consisting of two parts:

$$\text{isApplicable(application)} \Rightarrow \text{isValid(application)}$$

The first part determines whether a given validation rule is applicable to the insurance application data or not. If a rule is applicable, the insurance application must not violate the rule, i.e. isValid(application). Otherwise, the validation rule is ignored.

Because details of the case are confidential, a generic example is given to provide further illustration of validation rules. The example depicts two validation rules to define maximum sums that can be insured depending on the permissions of the insurance agents. The first validation rule is applicable to all applications created by insurance agents with the highest level of permission. The second validation rule is applicable to all applications that are created by insurance agents with lower permission level.

The distinction between the two validation rules is made by the first part of the implication:

Rule 1: $\texttt{isApplicable}(\texttt{application}):$

$$\texttt{application.agent.permission} = \texttt{highest\_level}$$

Rule 2: $\texttt{isApplicable}(\texttt{application}):$

$$\texttt{application.agent.permission} \neq \texttt{highest\_level}$$

The second part of the implication is used to enforce the maximum insured sum. As an application may consist of several partial contracts, the individual insured sums of all partial contracts are collected first. Afterwards, it is checked whether the total sum exceeds the threshold. While the structure of both rule’s isValid() parts is the same, different values for the maximum_insured_sum constant are used:

$$\texttt{isValid}(\texttt{application}):$$

$$\texttt{total\_sum} = \sum_{\texttt{partial} \in \texttt{application}} \texttt{partial.insured\_sum}$$

$$\texttt{total\_sum} \leq \texttt{maximum\_insured\_sum}$$

This example shows that many parameters may be involved in a validation rule, that intermediate calculations may be required, and that intermediate calculations may be reused in different validation rules. Therefore, all validation rules should be tested thoroughly.

For this case study, we consider the current set of validation rules as correct and treat them as our specification. By browsing the source code repository, we have identified 13 changes that have been made to the validation rules in order to correct them. Each change documents a fault that existed previously but is fixed prior to release. Based on these 13 changes, we reconstructed 13 implementation versions of which each contains one fault.

The 13 faults can also be classified according to our robustness fault classification (cf. [7]). Five faults can only be detected by invalid test inputs, while eight faults can be detected by both valid and invalid test inputs. Two of these five faults can be classified as *faults in error-signaling*. To reveal them, invalid test inputs must trigger EH which responds with an incorrect error code. The other three faults can be classified as *faults in error-detection conditions*. The conditions are too weak and do not detect invalid test inputs. Hence, the SUT incorrectly continues with its normal behavior.

The remaining eight faults that can be detected by both valid and invalid test inputs. They are *faults in error-detection conditions*. Four of theses faults have conditions that are too strong and therefore incorrectly detect exception occurrences for valid test inputs. The other four faults have characteristics of being too weak and too strict at the same time because wrong parameters with similar characteristics are used in the exception condition. As a consequence, an invalid test input may not violate the condition (too weak) while a valid test input may not satisfy the condition (too strong).

4.2. Data Collection Procedure

Data collection refers to the measurement and calculation of metric values from test execution. Therefore, metrics are defined in this section. Furthermore, the modeling of the IPM and RIPM as well as the selection and execution of test inputs is described.

4.2.1. Metrics

The resources available from the software development project are not directly analyzed and compared. Instead, they are used to reconstruct the implementation versions for test execution and to create a RIPM and an IPM that represent variations of insurance application data.

Based on the RIPM and IPM, test inputs are selected using a CT and a CRT test selection strategy. Then, the test inputs are executed on the 13 reconstructed implementations to assess the effectiveness.

A common metric to assess the effectiveness is **fault detection effectiveness** (FDE) [11, 16]. A test suite T is denoted as *failing* for a test scenario SC if at least one of the test inputs $\tau \in T$ detects the fault in SC.

$$\text{failing}(T, SC) = \begin{cases} 1 \text{ if } \exists \tau \in T \text{ that fails for } SC \\ 0 \text{ otherwise} \end{cases}$$

Using the failing function, FDE is defined as the ratio between the number of test suites T of a test suite family T^* that fail for a test scenario SC and the number of all test suites in the family T^*. In this case study, the family of test suites contains 20 different variants. In other words, the FDE is based on 20 randomized test suites that all satisfy the same coverage criterion for the same IPM or RIPM. They all test the same test scenario.

$$\text{FDE}(T^*, SC) = \frac{\sum_{T \in T^*} \text{failing}(T, SC)}{|T^*|}$$

Further on, the **average fault detection effectiveness** (AFDE) denotes the average FDE over a family of test scenarios SC^*. In our case study, the family of test scenarios SC^* consists of the 13 reconstructed implementations. The AFDE represents the average effectiveness of CRT and CT equally distributed over the 13 faults.

$$\text{AFDE}(T^*, SC^*) = \frac{\sum_{SC \in SC^*} \text{FDE}(T^*, SC)}{|SC^*|}$$

4.2.2. Modeling of IPM and RIPM

Since the FDE and AFDE metrics highly depend on the quality of the RIPM and IPM, a systematic modeling approach is necessary. We model the IPM first and later extend it with error-constraints to get a RIPM.

The IPM is modeled iteratively for one validation rule at a time. In each iteration, parameters and values are added to ensure that test inputs with the following three characteristics can be detected: (1) test inputs that are not applicable; (2) test inputs that are applicable and valid; (3) test inputs that are applicable but not valid. In addition, some exclusion-constraints are introduced to ensure syntactic correctness of selected test inputs. The IPM is considered as complete once the IPM contains all parameters and values necessary to satisfy branch coverage of each validation rule.

For the RIPM, the modeling of additional error-constraints is required. The error-constraints are modeled iteratively and we add new or update existing ones until the separation of valid and strong invalid test inputs conforms to the responses of the SUT, i.e. the SUT returns SUCCESS for each valid test input and the SUT returns an error code for each strong invalid test input.

In total, the IPM and RIPM consist of 32 parameters and 106 values. Most parameters have two, three, or four values each. But two parameters have six values each and one parameter has even nine values. Three exclusion-constraints of which each restricts combinations of two parameters are required to ensure syntactical correctness of the insurance applications. Furthermore, the RIPM contains 31 error-constraints. 15 error-constraints annotate single values as invalid. The remaining 16 error-constraints annotate schemata with 2, 3, or 5 values.

The complete IPM and RIPM are described below in exponential notation. For parameters and values, x^y refers to y parameters with x values. For exclusion- and error-constraints, x^y refers to y constraints with x parameters.

$$\text{Parameters \& Values: } 9^1 6^2 5^1 4^8 3^8 2^{12}$$

$$\text{Exclusion-Constraints: } 2^3$$

$$\text{Error-Constraints: } 5^2 3^6 2^8 1^{15}$$

4.2.3. Selecting and Executing Test Inputs

After creating the IPM and RIPM, both models are used to select sets of test inputs. Since we compare CRT with CT, two different test selection strategies are used. ROBUSTA is used to select test inputs for the RIPM and IPOG-C is used to select test inputs for the IPM.

To compare the FDE and AFDE of CRT with CT, test suites that satisfy different coverage criteria are used. We apply IPOG-C to select test suites that satisfy t-wise relevant coverage for $t \in \{1, ..., 5\}$. Furthermore, we apply ROBUSTA to select test suites that satisfy t-wise valid coverage with $t \in \{1, ..., 3\}$ and that satisfy b-wise strong invalid coverage with $b \in \{0, 1\}$.

To reduce the effect of accidental fault detection caused by ordering, the order of parameters and values of the input parameter models is randomly reordered and 20 different model variants are used to select test suites for each coverage criteria.

Table 1 depicts the average sizes of test suites that satisfy the different coverage criteria. Since ROBUSTA encompasses two coverage criteria (t-wise valid coverage and b-wise strong invalid coverage), the test suites are considered separately and combined.

The largest test suite is selected by IPOG-C which is required to satisfy t-wise relevant coverage with $t = 5$ (15023.70 test inputs). The second-largest test suite is also selected by IPOG-C to satisfy t-wise relevant coverage with $t = 4$ (2813.45 test inputs). The third-largest test suite is selected by ROBUSTA and satisfies t-wise valid coverage with $t = 3$ and b-wise strong invalid coverage with $b = 1$ (2224.30 test inputs).

When comparing the test suite sizes of t-wise relevant coverage of IPOG-C with t-wise valid coverage of ROBUSTA, it can be seen that the error-constraints drastically reduce the number of valid test inputs.

After test input selection, the test suites are used to stimulate the SUT in 13 different versions. Therefore, the 13 reconstructed implementations of which each contains one fault are tested to determine which test suite is able to detect which fault. The results are discussed in the following section.

Table 1
Test suite sizes of test suites for different coverage criteria

Coverage Criteria	t	b	Size
t-wise relevant coverage	1	-	9.00
	2	-	68.10
	3	-	480.10
	4	-	2813.45
	5	-	15023.70
t-wise valid coverage	1	-	7.00
	2	-	48.30
	3	-	267.95
b-wise strong invalid coverage	-	0	301.00
	-	1	1956.35
t-wise valid coverage and *b*-wise strong invalid coverage	1	0	308.00
	1	1	1963.35
	2	0	349.30
	2	1	2004.65
	3	0	568.95
	3	1	2224.30

5. Results & Discussion

In this section, the case study results regarding the computed FDE AFDE values are reported and discussed.

5.1. Fault Detection Effectiveness

Table 2 lists the FDE values of all test suites families applied to all 13 implementations. For better readability, + is used to indicate an FDE value of 1.00. The faults nos. 1 to 8 can all be detected by both valid and invalid test inputs, while the faults nos. 9 to 13 can only be detected by invalid test inputs. Again, the shown FDE value is an average value for one test suite family with 20 different test suites that are created by randomizing the order of parameters and values before selecting test inputs. As an example, in the first row for fault no. 3, an FDE value of 0.05 means that one out of 20 test suites detected the fault at least once per test suite.

As can be observed, t-wise relevant coverage is not able to detect all faults reliably. The FDE values increase when testing strength t grows. But even with $t = 5$ (15023.70 test inputs), only 7 faults are detected reliably (FDE value of 1.00). Further on, fault no. 10 remains undetected (FDE value of 0) and faults nos. 9 and 13 are only detected by one out of 20 test suites (FDE value of 0.05).

The CRT coverage criteria are characterized by avoiding the invalid input masking effect. Since all invalid schemata are excluded by t-wise valid coverage, the faults

Table 2
FDE values for different coverage criteria

Coverage			FDE values for faults nos. 1 to 13													AFDE
Criteria	**t**	**b**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**values**
t-wise relevant coverage	1	-	0	0	0.05	0.05	0	0	0	0	0	0	0.25	0.05	0	0.03
	2	-	0.10	0.10	0.45	0.20	0.10	0	0	0	0	0	0.65	0.20	0	0.14
	3	-	0.75	0.75	+	+	0.65	0.05	0.10	0.05	0.05	0	+	0.65	0	0.47
	4	-	+	+	+	+	+	0.15	0.10	0.05	0	0	+	+	0	0.56
	5	-	+	+	+	+	+	0.50	0.35	0.15	0.05	0	+	+	0.05	0.62
t-wise valid coverage	1	-	0.75	0.75	+	+	0.50	0.50	+	0.80	0	0	0	0	0	0.48
	2	-	+	+	+	+	+	+	+	+	0	0	0	0	0	0.62
	3	-	+	+	+	+	+	+	+	+	0	0	0	0	0	0.62
b-wise strong invalid	-	0	+	+	+	+	+	+	0.90	0.80	+	+	+	+	+	0.98
	-	1	+	+	+	+	+	+	+	+	+	+	+	+	+	+
t-wise valid coverage and b-wise strong invalid coverage	1	0	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	1	1	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	2	0	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	2	1	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	3	0	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	3	1	+	+	+	+	+	+	+	+	+	+	+	+	+	+

nos. 9 to 13 cannot be detected. But for all other faults, *t*-wise valid coverage has higher FDE values for the same testing strength *t* when compared to *t*-wise relevant coverage. Because invalid input masking is avoided, a testing strength of $t = 2$ is sufficient to detect faults nos. 1 to 8 reliably (FDE values of 1.00).

Using *b*-wise strong invalid coverage with $b = 0$, 11 out of 13 faults can already be detected reliably and the two remaining faults have high FDE values of 0.90 and 0.80. The effectiveness of robustness interactions is even higher and all faults can be detected reliably with $b = 1$.

Four faults that have too strong error detection conditions and that actually require valid test inputs to be detected are also reliably detected by *b*-wise strong invalid coverage. We could observe that a strong invalid test input that is expected to violate the error detection condition of the *l*-th validation rule is also expected to satisfy all prior validation rules from 1 to $l - 1$. Therefore, strong invalid test inputs can be considered as "partially-valid" test inputs that are able to accidentally detect faults that require valid test inputs. This effect is strengthened by robustness interactions because more test inputs are selected and more interactions are covered by them.

ROBUSTA combines *t*-wise valid coverage and *b*-wise strong invalid coverage and the FDE values show that test suites for both coverage criteria complement each other. Since valid and strong invalid test inputs are able to detect faults nos. 1 to 8, the FDE values are complemented by the combination of both test suites. For faults nos. 9 to 13, the FDE values are not complemented by the combination of both test suites. This is because test suites that only satisfy *t*-wise valid coverage cannot detect these faults. Therefore, the FDE values of the combined test suites are the same as the FDE values of the test suites that satisfy *b*-wise strong invalid coverage.

In order to detect all faults reliably, the *b*-wise strong invalid coverage must be selected because faults nos. 9 to 13 remain undetected otherwise. Either robustness interaction ($b > 0$) or the combination of *b*-wise strong invalid coverage with *t*-wise valid coverage is required to reliably detect faults nos. 1 to 8. Even though $t = 1$ is only sufficient to detect three of the first eight faults reliably, the combination with *b*-wise strong invalid coverage improves the FDE and all faults can be detected reliably.

The discussion of the FDE shows which coverage criteria are appropriate to reliably detect different types of faults. Next, we discuss the AFDE over all 13 faults.

5.2. Average Fault Detection Effectiveness

Because AFDE values are average values over a set of faults, AFDE allows making general statements about both the effectiveness and the efficiency of coverage criteria. First, we discuss the effectiveness in terms of AFDE values of different coverage criteria. Therefore, Table 2 lists the AFDE values for test suites that satisfy different coverage criteria. Afterwards, we discuss the efficiency in terms of AFDE values in relation to test suite sizes (listed in Table 1).

The AFDE values reflect what we discussed before since they aggregate FDE values. Because of the invalid input masking effect, test suites that satisfy *t*-wise relevant coverage only reach an AFDE value of 0.62.

In direct comparison, test suites that satisfy *t*-wise valid coverage reach a maximum AFDE value of 0.62 as well. The same AFDE value can be reached because they prevent invalid input masking. However, the AFDE value cannot be further improved by increasing the testing strength because faults nos. 1 to 8 are already detected reliably and faults nos. 9 to 13 cannot be detected by valid test inputs. Comparing the two coverage criteria for each testing strength individually shows that the AFDE value of *t*-wise valid coverage is always higher than the AFDE value of *t*-wise relevant coverage.

For *b*-wise strong invalid coverage, the lowest AFDE value is 0.92 (no robustness interactions) which is always higher than the AFDE values of *t*-wise relevant and valid coverage. Furthermore, *b*-wise strong invalid coverage with robustness interactions has an AFDE value of 1 and therefore detects all faults reliably.

Overall, the combination of *t*-wise valid coverage and *b*-wise strong invalid coverage performs the best and always detects all faults reliably.

When putting the AFDE values in relation to test suite sizes, it can be noted that *t*-wise relevant coverage has the worst efficiency as it requires 15023.70 test inputs for an AFDE value of 0.62. In contrast, *t*-wise valid coverage only requires 48.30 test inputs for an AFDE value of 0.62.

The best efficiency is offered by the combination of *t*-wise valid coverage with $t = 1$ and *b*-wise strong invalid coverage with $b = 0$ which requires 308.00 test inputs for an AFDE value of 1.00. When using an AFDE value of 0.92 as a lower boundary (12 out of 13 faults), *b*-wise strong invalid coverage with $b = 0$ is sufficient and only requires 301.00 test inputs for an AFDE value of 0.98.

This discussion about efficiency is, of course, influenced by the characteristics of the 13 faults and cannot be generalized. But as more general statements, it can be observed that *t*-wise relevant coverage requires more test inputs to reach a similar AFDE value than *t*-wise valid coverage, *b*-wise strong invalid coverage, or the combination of both. At the same time, the combination of *t*-wise valid coverage and *b*-wise strong invalid coverage always has an AFDE value of 1.00 while at most 2224.30 test inputs are used. This finding is also consistent with our prior experimental evaluation (cf. [7]).

Therefore, we draw the conclusion that *t*-wise valid coverage, *b*-wise strong invalid coverage, and the combination of both perform as well as or better than *t*-wise relevant coverage in terms of effectiveness and efficiency. Although, the findings are only derived from one particular case. Therefore, we do not consider this to be true for all SUTs but for SUTs with many validation rules.

6. Threats to Validity

We compare the effectiveness of CRT using an implementation of the ROBUSTA test selection strategy with CT using an implementation of the IPOG-C test selection strategy. To ensure an unbiased implementation, both implementations follow the guidelines of Kleine & Simos [20]. Further on, the source code of the test selection strategies is published as part of the `coffee4j` open-source test automation framework[1].

The effectiveness of CRT and CT highly depend on the IPM and RIPM. Furthermore, the effectiveness depends on the faults that are considered in this case study.

Unfortunately, details of the case, i.e. source code of the validation rules and detailed descriptions of the faults, are confidential. To improve transparency and reproducibility, we describe the faults and make the characteristics of the IPM and RIPM explicit.

To avoid any bias, both the IPM and RIPM are modeled systematically and share the same set of parameters and values. To prevent falsified results due to accidental fault triggering, the orders of parameters and values are randomized and 20 different variants are used in test input selection. All presented FDE values are average values.

Since this is a case study with only one case, it is difficult to generalize the findings [10]. Further on, it has to be noted that the archival data of this case study is only a snapshot and the ground truth, i.e. the existing and previously existing faults, is unknown. Hence, the data can be biased towards simpler faults that are easier to detect. To prevent too far-reaching conclusions, we describe the characteristics of the SUT and also limit our conclusions to similar systems with many validation rules.

7. Conclusion

CRT extends CT to generate separate test suites with valid and strong invalid test inputs in order to avoid input masking that is caused by EH. Therefore, CRT requires additional effort to model error-constraints and introduces additional complexity to test selection strategies because error-constraints must be considered. This raises the question about the usefulness of CRT and whether the avoidance of input masking outweighs the additional effort and complexity. Until now, only artificial test scenarios are used to compare CT with CRT and it remains unclear if indicated advantages of CRT can be transferred to real-world scenarios.

In this paper, we therefore present the results of a case study based on a real-world system with 31 validation rules and 13 previously existing faults. To compare CT with CRT, we construct a IPM and a RIPM, select test inputs, and stimulate 13 implementations of the real-world system of which each implementation contains one of the 13 previously existing faults. For the subsequent discussion, we introduce the FDE and AFDE metrics.

To summarize the findings of this case study, we discuss both research questions individually.

Research Question 1: Our results indicate that the CRT test method is applicable in real-world test scenarios. This case study demonstrated that RIPMs with 32

[1] See https://coffee4j.github.io for more information.

parameters and 31 error-constraints can be constructed. Further on, the ROBUSTA test selection strategy is capable of selecting test suites for RIPMs with 32 parameters and 31 error-constraints.

Research Question 2: The comparison of CRT with CT is consistent with the findings of our previously conducted controlled experiment with artificial test scenarios (cf. [7]). Since the case under analysis has much EH, CRT performs better than CT in terms of FDE. Further on, it requires fewer test inputs to achieve better AFDE values than CT.

Therefore, we draw the conclusion that *t*-wise valid coverage, *b*-wise strong invalid coverage, and the combination of both perform as well as or better than *t*-wise relevant coverage in terms of effectiveness and efficiency.

Although, the FDE and AFDE values are influenced by the characteristics of the 13 faults and cannot be generalized. Therefore, we do not consider this to be true for all SUTs but for SUTs with much EH.

In future work, we plan to conduct further case studies to learn more about the FDE of CRT and CT.

References

[1] IEEE, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 (1990).

[2] A. Avižienis, J. Laprie, B. Randell, C. E. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Sec. Comput. 1 (2004) 11–33.

[3] C. Marinescu, Are the classes that use exceptions defect prone?, in: Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, EVOL/IWPSE 2011, Szeged, Hungary, September 5-6, 2011., 2011, pp. 56–60.

[4] P. Sawadpong, E. B. Allen, B. J. Williams, Exception handling defects: An empirical study, in: 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering, 2012, pp. 90–97.

[5] C. Nie, H. Leung, A survey of combinatorial testing, ACM Comput. Surv. 43 (2011) 11:1–11:29.

[6] G. B. Sherwood, Effective testing of factor combinations, in: Proceedings of the Third International Conference on Software Testing, Analysis and Review, Washington, DC, 1994, pp. 151–166.

[7] K. Fögen, H. Lichter, Combinatorial robustness testing with negative test cases, in: Proceedings of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019, Sofia, Bulgaria, July 22-26, 2019, 2019, pp. 34–45.

[8] K. Fögen, H. Lichter, An experiment to compare combinatorial testing in the presence of invalid values, in: Proceedings of the 7th International Workshop on Quantitative Approaches to Software Quality co-located with 26th Asia-Pacific Software Engineering Conference (APSEC 2019), Putrajaya, Malaysia, December 2, 2019., 2019, pp. 27–36.

[9] B. A. Kitchenham, L. Pickard, S. L. Pfleeger, Case studies for method and tool evaluation, IEEE Softw. 12 (1995) 52–62.

[10] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2009) 131–164.

[11] M. Grindal, J. Offutt, S. F. Andler, Combination testing strategies: a survey, Softw. Test., Verif. Reliab. 15 (2005) 167–199.

[12] C. Nie, H. Leung, The minimal failure-causing schema of combinatorial testing, ACM Trans. Softw. Eng. Methodol. 20 (2011) 15:1–15:38.

[13] L. Yu, Y. Lei, M. N. Borazjany, R. Kacker, D. R. Kuhn, An efficient algorithm for constraint handling in combinatorial test generation, in: Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013, 2013, pp. 242–251.

[14] D. M. Cohen, S. R. Dalal, M. L. Fredman, G. C. Patton, The AETG system: An approach to testing based on combinatiorial design, IEEE Trans. Software Eng. 23 (1997) 437–444.

[15] J. Czerwonka, Pairwise testing in real world, in: 24th Pacific Northwest Software Quality Conference, volume 200, Citeseer, 2006.

[16] J. Petke, M. B. Cohen, M. Harman, S. Yoo, Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection, IEEE Trans. Software Eng. 41 (2015) 901–924.

[17] H. Wu, n. changhai, J. Petke, Y. Jia, M. Harman, An empirical comparison of combinatorial testing, random testing and adaptive random testing, IEEE Transactions on Software Engineering (2018) 1–1.

[18] K. Fögen, H. Lichter, A case study on robustness fault characteristics for combinatorial testing - results and challenges, in: Proceedings of the 6th International Workshop on Quantitative Approaches to Software Quality co-located with 25th Asia-Pacific Software Engineering Conference (APSEC 2018), Nara, Japan, December 4, 2018., 2018, pp. 22–29.

[19] P. Wojciak, R. Tzoref-Brill, System level combinatorial testing in practice - the concurrent maintenance case study, in: Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014, March 31 2014-April 4, 2014, Cleveland, Ohio, USA, 2014, pp. 103–112.

[20] K. Kleine, D. E. Simos, An efficient design and implementation of the in-parameter-order algorithm, Mathematics in Computer Science 12 (2018) 51–67.