An Experiment to Compare Combinatorial Testing in the Presence of Invalid Values

Konrad Fögen  
Research Group Software Construction  
RWTH Aachen University  
Aachen, NRW, Germany  
foegen@swc.rwth-aachen.de

Horst Lichter  
Research Group Software Construction  
RWTH Aachen University  
Aachen, NRW, Germany  
lichter@swc.rwth-aachen.de

Abstract—Robustness is an important property of software that should be thoroughly tested. Combinatorial testing (CT) is an effective black-box test approach. When using it for robustness testing, the input masking effect can prevent faults from being detected. However, the impact is not yet clear. Therefore, we conducted a controlled experiment to understand how input masking affects the fault detection effectiveness of CT and how effective CT is in the presence of error-handling and invalid values.

Keywords—Software Testing, Combinatorial Testing, Robustness

I. INTRODUCTION

Robustness is an important property of software systems that describes “the degree to which a system or component can function correctly” in the presence of external faults like invalid inputs [1]. External faults can have a severe impact on the system’s robustness because they can propagate to system failures resulting in abnormal behavior or system crashes [2]. To improve robustness, systems implement error-handling to appropriately react to external faults [3]. Oftentimes, the external fault cannot be resolved by the system internally [4]. Then, the system is terminated by the error-handling procedure that returns an error-message to the client without executing the normal procedure. This is also referred to as the error-propagation strategy [5]. Unfortunately, error-handling procedures have a fault density that is up to three times higher compared with normal procedures [6]. Therefore, testing is important to check error-handling.

The purpose of testing is to reveal failures by stimulating a system under test (SUT) with test inputs and observing the results via test oracles [2]. To reveal a failure, a fault must be triggered to produce an error and the error must propagate to a failure of the SUT [7]. Assuming that test oracles reveal all propagated failures, the important factor in testing is the selection of test inputs such that a fault is triggered.

Combinatorial testing (CT) is a black-box approach for test input selection [8]. A test model describes the SUT via input parameters and input values. Using a combination strategy, test inputs are selected from the test model such that they satisfy a combinatorial coverage criterion like $t$-wise that is satisfied if all value combinations of $t$ parameters appear in at least one test input.

Combinatorial robustness testing (CRT) is an extension to CT that incorporates robustness testing [5]. CRT is necessary because of the input masking effect: The first invalid value that is evaluated by the SUT triggers error-handling and the normal control-flow is abandoned. When the error-propagation strategy is used, the SUT returns with an error-message and the normal control-flow is not resumed. Then, the other values and value combinations of the test input remain untested because they are masked.

CRT avoids input masking by separating the testing with valid test inputs, that do not contain any invalid value, from testing with strong invalid test inputs, that contain exactly one invalid value. Previous experiments [5] have shown that CRT is an effective approach for combinatorial testing in the presence of error-handling. However, in comparison to CT, CRT requires additional semantic information about invalid values. CT can also reveal failures in the presence of error-handling without requiring additional information. But, the input masking effect can prevent faults from being triggered.

Therefore, our aim is to answer the following research question: How effective is CT in triggering faults when error-handling and invalid values are present? To answer the research question, we apply the $t$-factor fault model, derive influencing factors and conduct a controlled experiment.

The paper is structured as follows. First, an example is introduced. Then, Section III and IV summarize background and related work. In Section V, the $t$-factor fault model is applied to the context of error-handling and factors that influence fault triggering are identified. Afterwards, the experiment design is discussed in Section VI and the results are presented in Section VII. Afterwards, threads to validity are discussed and we conclude with a summary of our work.

II. EXAMPLE

To illustrate the impact of error-handling, we reuse an example from previous work [5]. The example is a customer registration service with three validity checks to ensure that the entered data is not invalid. Since the service cannot correct the data itself, an error code is returned to the client asking to correct the data.

A test model for CT is depicted in Figure 1 with 123 representing some invalid value. Each test input that contains
an invalid value like [Title:123] should yield an error code. However, assuming the implementation of the service contains a fault in the validity check for given names that returns a wrong error code (address instead of name error). It is triggered whenever an invalid given name, e.g., [GivenName:123], is evaluated. An implementation is illustrated in Listing 1 with INV_xxx string literals representing specific error codes.

String register(String title, given, address){
    if(isInvTitle(title)) return INV_TITLE;
    if(isInvGivenName(given)) return INV_ADDRESS;
    if(isInvAddress(address)) return INV_ADDRESS;
    ...
}  
Listing 1. Example of an Input Validity Check

A test input like [Title:Mrs, GivenName:123, Address:UK] would trigger the fault. In contrast, a test input like [Title:123, GivenName:123, Address:UK] would yield INV_TITLE because of the invalid title. But, it would not trigger the name check fault because of input masking.

To satisfy 1-wise coverage, a minimal set of three test inputs is generated with exactly one test input that contains [GivenName:123]. In total, nine test inputs with [GivenName:123] exist and the combination strategy must select one. Of the nine test inputs, three contain [Title:123] causing input masking. Therefore, the probability of triggering the fault is \( \frac{3}{9} = 0.66\% \).

Increasing the testing strength to \( t > 1 \) will also increase the fault triggering probability. However, finding a minimal set of test inputs that satisfies \( t \)-wise coverage for \( t \geq 2 \) is in general NP hard [9]. Heuristics are used as combination strategies which produce small but not always minimal sets of test inputs. Then, some \( t \)-sized value combinations appear more than once and the probability cannot be simply calculated.

III. BACKGROUND

A. Combinatorial Testing

CT is a well-known approach to black-box testing where test inputs are selected based on a test model [8]. The test model \( TM \) describes the input space of a program as a set of \( n \) parameters \( P = \{ p_1, \ldots, p_n \} \) and the domain of each parameter \( p_i \) is a finite nonempty set of \( m_i \) values \( V_i = \{ v_{i1}, \ldots, v_{im_i} \} \). A combination \( \tau \) is a set of \( 0 < d \leq n \) parameter-value pairs \( (p_i, v_j) \) for \( d \) distinct parameters \( p_i \) with \( v_j \in V_i \). A test input is a combination of size \( d = n \). Combination \( \tau_a \) covers another combination \( \tau_b \) if every parameter-value pair of \( \tau_b \) is included in \( \tau_a \), which we denote as \( \tau_b \subseteq \tau_a \).

In CT, coverage criteria and combination strategies depend on a specific test model \( TM \) [8]. A coverage criterion \( C \) describes requirements of \( TM \) that must be met by a set of test inputs \( T \subseteq D \) and a combination strategy describes how to select test inputs \( T \subseteq D \) such that \( C \) is satisfied.

The \( t \)-wise coverage criterion is a common criterion that is satisfied if all value combinations of \( t \) parameters appear in at least one test input [8]. In addition, all smaller combinations \( (d < t) \) are covered and also some larger combinations of size \( t' = (t + k) \) with \( k > 0 \) and \( t' \leq n \) are covered [10]. This so called collateral coverage can potentially help triggering additional faults [11].

The input domains of real-world systems are typically restricted. As a consequence, certain values or value combinations are not of any interest or may prevent a test from being executed. Exclusion-constraints are commonly used to exclude irrelevant value combinations from test input selection [5]. Every test input that satisfies the exclusion-constraints is a relevant test input.

B. Combinatorial Robustness Testing

CRT is an extension to CT that incorporates robustness testing [5]. It explicitly considers the input masking effect that is caused by error-handling. To avoid input masking, CRT separates the generation of valid and invalid test inputs.

Therefore, additional semantic information is required to model invalid values [5]. The additional information can be modelled via error-constraints which denote a second set of constraints [5]. Then, relevant test inputs are further partitioned as follows. A relevant test input is a valid test input if it satisfies all exclusion- and all error-constraints. A relevant test input is invalid if it satisfies all exclusion-constraints but at least one error-constraints remains unsatisfied. An invalid test input is denoted a strong invalid test input if exactly one error-constraint is unsatisified.

Then, valid test inputs and invalid test inputs are generated separately such that they satisfy different coverage criteria. The valid \( t \)-wise coverage criterion is satisfied if each valid parameter value combination of size \( t \) appears in at least one test input \( \tau \) of which all other values and value combinations are also valid [5], [8]. In addition, the single error coverage criterion is satisfied if each invalid value appears in at least one strong invalid test input \( \tau \) of which all other values and value combinations are valid [5], [8].

When satisfying the aforementioned coverage criteria, both normal procedures and error-handling are tested without input masking. However, in comparison to CT, additional effort is required to model the error-constraints.

IV. RELATED WORK

Cohen et al. [12] first described the input masking effect caused by error-handling [8]. They also noted the need to separate valid and invalid test inputs to avoid input masking. An evaluation of combination strategies by Grindal et al. [13] discussed an example where input masking prevented a fault from being triggered. A case study by Wojciak and Tzoref-Brill [14] reported on CT including testing with invalid inputs.
The CT tools AETG [12], ACTS [15] and PICT [16] allow to mark individual values as invalid and also generate separate sets of test inputs. However, invalid value combinations are not directly supported.

In previous work [5], [17], we introduced error-constraints as a modelling technique that allows to directly model both invalid values and invalid value combinations. We also conducted experiments. But, they focused on the interaction between invalid and valid parameter combinations. Furthermore, we discussed techniques to identify and explain over-constrained test models [18] and discussed a technique to semi-automatically repair over-constrained test models [19].

In a case study [20], we analyzed bug reports of a software for life insurances. 51 out of 212 analyzed bug reports describe robustness faults. Many of them were triggered by invalid value combinations and we concluded that it is not sufficient for a CT tool to only consider invalid values.

Despite the conclusion, we only consider invalid values in this experiment because it allows a clearer separation between valid and invalid values when extending a given test scenario.

41 robustness faults were triggered by a single invalid value nine robustness faults were triggered by an invalid combination of two values and one fault was triggered by an invalid combination of three values.

Empirical studies also compared the efficiency of CT with random testing. A recent study summarizes previous comparisons [21]. However, no study focused on error-handling and one excluded error-handling explicitly (See [11]).

V. APPLYING THE T-FACTOR FAULT MODEL IN THE PRESENCE OF ERROR-HANDLING

A. Overview

The idea of the t-wise coverage criterion is based on the corresponding t-factor fault model which is formally introduced by Dalal and Mallows [22]. In general, a fault model is a description of hypothesized faults. The t-factor fault model relies on a transformational model of the SUT where the output is defined in terms of its input [23]. The input is modelled as a set of parameters $p_1, \ldots, p_n$ with each parameter $p_i$ having a domain $D_i$ potentially infinite of values.

The faults are also defined in terms of the SUT’s input. It is assumed that faults are caused by the interaction of $t$ parameters and a t-factor fault is triggered by a combination of $t$ parameter values. A t-factor fault can be described by a condition over $t$ parameters which must be satisfied by an input to the SUT in order to trigger the fault. Each input that satisfies the condition triggers the t-factor fault.

The t-factor fault model is researched empirically [20], [24]–[29] where bug reports for different types of software are analyzed. An interaction rule is derived from the empirical findings which states that “only a few factors are involved in failure-inducing faults in software. Most failures are induced by single factor faults or by the interaction of two factors; progressively fewer failures are induced by interactions between three, four, or more factors. The maximum degree of interaction in actual faults so far observed is six” [30].

Since the t-wise coverage criterion is defined relative to the test model, the SUT model and test model share the same set of input parameters. While the domain $D_i$ of a SUT input parameter $p_i$ is potentially infinite, the domain $V_i \subseteq D_i$ of a test model parameter $p_i$ is a finite nonempty subset of values that contains all values a tester is interested in.

When testing with a test suite that satisfies the t-wise coverage criterion, all parameter value combinations of size $t$ appear in some test input. Testing should fail for each SUT that contains $t'$-factor faults with $t' \leq t$ if the values of the test model are selected properly such that the condition of the $t'$-factor faults can be satisfied. Therefore, CT is also called pseudo-exhaustive testing implying that t-wise testing is as good as exhaustive testing for a particular class of software with faults of factor $t$ or smaller [26].

A test input $\tau$ that triggers a t-factor fault contains a combination $c$ that is a failure-inducing combination (FIC). Each test input $\tau$ that covers $c \subseteq \tau$ triggers a fault [31]. A FIC $c$ is minimal (MFIC) if no proper subset $d' \subset c$ triggers a fault. The size of a MFIC is predetermined by the t-factor fault and its condition.

When applying the t-factor fault model to faults in the presence of error-handling, characteristics that affect the capability of triggering faults can be derived. The characteristics either affect the t-factor faults or the FICs that trigger them. They are discussed in the following two subsections.

Recall the fault of the example implementation (Figure 1) that is triggered whenever an invalid given name is evaluated. To describe it as a t-factor fault, the error-handling must be taken into account. It is not a 1-factor fault because not every input that satisfies \texttt{isInvGivenName(given)} triggers the fault. Consequently, \texttt{[GivenName:123]} is not a FIC. Triggering the fault requires a valid title because the error-handling propagates INV\_TITLE otherwise. Therefore, the fault can be modelled as a 2-factor fault using a conjunction over title and given: $\neg \texttt{isInvTitle(title)} \land (\texttt{isInvGivenName(given)}).

For the given test model, the combinations [Title:Mr, GivenName:123] and [Title:Mrs, GivenName:123] are minimal failure-causing.

B. Characteristics affecting Size of t-Factor Faults

1) Number of Parameters involved in Error-Handling: In the presence of error-handling, the condition to trigger a fault can be formulated as a conjunction of two sub-conditions. First, the location of an incorrect error-handler must be reached [7] by ensuring that no prior error-handler terminates the SUT. We denote this as the prevention sub-condition. Second, an invalid value or invalid value combination must cause an error-handler to produce an incorrect program state (infection [7]) that can propagate to a failure. We denote this as the infection sub-condition.

For the example, $\neg \texttt{isInvTitle(title)}$ is used for prevention and $\texttt{(isInvGivenName(given))}$ is used for infection.
The size of a $t$-factor fault increases with the number of parameters involved in prevention and infection sub-conditions. To guarantee that a $t$-factor fault is triggered, the test input set must satisfy $t$-wise coverage of the same size.

2) **Priority of Error-Handlers:** Each error-handler with an earlier position in the control-flow has the potential to terminate the SUT before the incorrect error-handler is reached. Therefore, each prior error-handler increases the prevention sub-conditions.

A fault in the first error-handler of the example would be modelled by an empty prevention sub-condition. In contrast, a fault in the third error-handler would be modelled by a prevention sub-condition that includes both prior error-handlers, e.g. $\neg(isInvTitle(title)) \lor isInvGivenName(given)$.

However, the modelled fault depends on a specific implementation whereas CT is a black-box approach which depends on the SUT’s specification instead. While testing with 2-wise coverage is sufficient to detect a fault in error-handling that can be modelled as a 2-factor fault, it requires the location of the incorrect error-handler to be known beforehand in order to determine the appropriate testing strength of $t = 2$.

The specification does typically not impose a specific order of error-handling. For instance, an implementation that checks the validity of the address first is as correct as the implementation shown in Listing 1 where the address is checked last. Then, the fault in the validity check of the given name becomes a 3-factor fault.

To make the $t$-factor fault model and the determination of testing strength $t$ independent from the location of the incorrect validity check within the control-flow, all error-handlers in every possible order must be taken into account. Then, $t$ would grow with the number of parameters checked by error-handlers.

Using a testing strength of $t = 3$ ensures that the fault is triggered for all possible orderings of the error-handlers. Thereby, the prior knowledge about the incorrect error-handler is also abandoned. By deriving the testing strength from all parameters that are involved in any error-detection condition, it is ensured that each error-handler is reached and potential faults are triggered.

While this testing strength denotes the lower limit to ensure that potential faults are triggered independently from the ordering of error-handlers, testing is still conducted for a specific implementation with a specific ordering. Therefore, we distinguish the effective prevention sub-condition which is sufficient for a specific implementation from the general prevention sub-condition that is sufficient for all orderings. On average, the effective prevention sub-condition considers fewer error-handlers which improves the likelihood of triggering a fault when using a testing strength that is not sufficient for the general prevention sub-condition.

C. **Characteristics affecting the Number of Minimal Failure-inducing Combinations**

1) **Number of Valid Values:** Given a test model and a $t$-factor fault $f$, a parameter $p$ is involved if the condition that describes $f$ includes $p$. Otherwise, $p$ is not involved.

For the example, parameters Title and GivenName are involved for the fault with the condition $\neg(isInvTitle(title)) \land (isInvGivenName(given))$ while parameter Address is not involved.

In the example, two MFICs of size $t = 2$ exist that trigger the same fault. A test suite that satisfies 2-wise coverage includes each MFIC at least once and guarantees that the fault is triggered. Since two MFICs trigger the same fault, the probability of selecting one of them is increased when testing with $(t' < 2)$-wise collateral coverage. A set of test inputs that only satisfies 1-wise coverage has a probability of at least $\frac{2}{t'} = 0.66\%$ to trigger the fault, i.e. at least one test input covers [GivenName:123] and there is a $\frac{1}{t'}$ chance that [Title:123] is not covered by the same test input.

The effect of values can be distinguished depending on whether or not the value’s parameter is involved in the infection or prevention sub-condition.

A valid value or valid value combination that is involved in the infection sub-condition does not affect the failure-inducing combinations because satisfying the infection sub-condition and being valid are mutually exclusive. For instance, adding another valid given name [GivenName:Jim] to the example test model does not affect the FICs because it cannot satisfy (isInvGivenName(given)).

But, valid values and valid value combinations of parameters that are involved in the prevention sub-condition increase the number of MFICs. For instance, adding another valid value [Title:Sir] to the example results in another MFIC [Title:Sir, GivenName:123]. For 1-wise testing, the probability to trigger the fault increases to $\frac{3}{t'} = 0.75\%$.

Valid values and valid value combinations of parameters that are not involved in the prevention and infection sub-condition of a $t$-factor fault do not directly affect the MFICs. They contribute to the set of $t$-sized parameter values combinations that must be covered by some test input to satisfy $t$-wise coverage, though.

For our example, nine test inputs can be created that cover [GivenName:123], i.e. $\{Mr, Mrs, 123\} \times \{UK, US, 123\}$. Since three of them cover [Title:123] and do not satisfy the effective prevention sub-condition, the probability of selecting a test input that covers a MFIC are $\frac{3}{9} = 0.66\%$. When another valid address is added, 12 test inputs that cover [GivenName:123] can be created and four of them do not satisfy the effective prevention sub-condition. The probability of triggering the fault remains the same.

It has to be noted that additional values can have an effect on the overall selection of test inputs. Maybe a minimal test suite does not exist for the given values or another reason that is inherent to the combination strategy increases or decreases redundancy and thus affects the fault triggering probability. However, these effects are beyond the scope of this paper.

2) **Number of Invalid Values:** Invalid values of parameters that are involved in the condition of a $t$-factor fault $f$ can increase or decrease the probability of triggering $f$. It depends on whether the parameters are involved in the prevention or
infection sub-condition.

When the parameter of the invalid value is involved in the prevention sub-condition, the probability of input masking is increased. For the example, adding another invalid value [Title:456] decreases the probability of selecting a test input that covers one of the two MFICs to $\frac{1}{2} = 0.50\%$.

When the parameter of the invalid value is involved in the infection sub-condition, the probability of input masking is decreased. For instance, adding another invalid value [GivenName:456] that triggers the same fault as [Address:123] adds two more MFICs [Title:Mr, GivenName:456] and [Title:Mrs, GivenName:456] to the example.

Invalid values of parameters that are not involved in $t$-factor fault $f$ only exist for effective prevention sub-conditions because general prevention sub-conditions consider all error-handlers. As an example, consider an additional invalid address [Address:456].

From the perspective of general prevention sub-conditions, they can be treated as above invalid values that increase the probability of input masking.

From the perspective of effective prevention sub-conditions, the invalid values do not affect the MFICs because the evaluation would happen after the evaluation of the incorrect error-handler.

VI. EXPERIMENT DESIGN
A. Test Scenarios

The objective of our experiment is to evaluate the effectiveness of CT when generating test inputs in the presence of error-handling. Especially, we want to determine in which cases CT is sufficient enough such that CRT and its additional effort can be avoided. Therefore, we generated test inputs with different characteristics and executed them in test scenarios.

The illustrated test scenario uses 3 parameters where the second error-handler (index $i = 1$) is incorrect. The number of valid and invalid values per parameter is implicitly encoded by the test model that is used to generate test inputs. By varying the index of the incorrect error-handler, three different test scenarios for our example can be created, where either the first, second or third error-handler is incorrect. A set of test scenarios, that shares the same parameters and values but differs in the index of the incorrect error-handler is called a test scenario family $S^*$. As a notation, we use $P-V-I$ where $P$ refers to the number of parameters involved in error-handling, $V$ refers to the number of valid values per parameter, and $I$ refers to the number of invalid values per parameter.

The experiment starts with a root test scenario family 6-2-1 representing a simple application of CT. The root test scenario family is then extended such that the number of parameters involved in error-handling, the valid values, and the invalid values per parameter are increased. In total, the following test scenario families are used in the experiment: 6-2-1, 6-5-1, 6-2-4, 12-2-1, 12-5-1, 12-2-4, 18-2-1, 18-5-1, 18-2-4.

B. Test Input Generation

The test models used in this experiment share the same set of parameters with the test scenario and define the number of valid and invalid values per parameter.

Given a test model and a testing strength $t$, a combination strategy generates a test suite $T$. To avoid any bias, we use test suites from a publicly available repository (see [32]) that contains many of the smallest known test suites. They are generated with the IPOG-F combination strategy that is also implemented in ACTS [15].

The order of parameters and values in a test model has no impact on whether or not a generated test suite satisfies $t$-wise coverage. However, it has an impact on which $t$-wise parameter value combinations are combined in a single test input. To reduce the effect of accidental fault triggering that is caused by ordering, the parameters and values of a test suite are randomly reordered and 50 different variants of each test

---

1https://will be published after review
suite are generated. The set of all test suite variants is called a test suite family $T^*$. The testing strengths used in the experiment range from $t = 1$ to $t = 5$ because most failures are induced by this range, according to the interaction rule.

### C. Evaluation Metrics

A common metric to evaluate combination strategies is called Fault Detection Effectiveness (FDE) \cite{8}, \cite{11}.

A test suite $T$ is denoted as failing for a test scenario $S$ if at least one of the test inputs $r \in T$ triggers the $t$-factor fault $f \in S$ and the test suite consequently fails.

$$\text{failing}(T, S) = \begin{cases} 1 & \text{if } \exists r \in T \text{ that fails for } S \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Using the failing function, FDE is defined as the ratio between the number of test suites $T \subseteq T^*$ of a family that fail for a test scenario $S$ and the number of all test suites in a family $T^*$ that is used to test $S$.

$$\text{FDE}(T^*, S) = \frac{\sum_{T \in T^*} \text{failing}(T, S)}{|T^*|} \quad (2)$$

In other words, the FDE is based on randomized variants of a test suite that all satisfy the same testing strength. They all are used to test the same test scenario $S$ which has a fixed incorrect error-handler. While this metric can be used to identify characteristics that may influence the FDE, the information cannot be used in practice because one must know which error-handler is incorrect.

Therefore, we introduce the average fault detection effectiveness (AFDE) which is the average FDE over a family of test scenarios $S^*$. Thus, AFDE represents the effectiveness of a test scenario family when knowing that one error-handler is incorrect but without knowing its index.

$$\text{AFDE}(T^*, S^*) = \frac{\sum_{S \in S^*} \text{FDE}(T^*, S)}{|S^*|} \quad (3)$$

### VII. Results & Discussion

#### A. Overview

The overall results of the experiment are consistent with the application of the $t$-factor fault model. Whenever the first error-handler is incorrect, the prevention sub-condition is empty and one parameter is involved in the infection sub-condition. The resulting $t$-factor fault is triggered in each test scenario by all test suites that satisfy the testing strength $t = 1$. Whenever an error-handler at a higher index is incorrect, the prevention sub-condition includes the parameters checked by all error-handlers with lower indices. Since one parameter is involved in the infection sub-condition, the faults can be described as $(\text{index} + 1)$-factor faults, where the index of the first error-handler is 0. For all considered testing strengths $(1 \leq \tau \leq 5)$, all $(\text{index} + 1)$-factor faults are triggered by all test suites that satisfy the corresponding testing strength.

Beyond that, collateral coverage results in $(\text{index} + 1)$-factor faults are repeatedly triggered by test suites that only satisfy lower testing strengths.

#### B. Fault Detection Effectiveness

Table I depicts an excerpt of FDEs computed for the $6-2-1$ test scenario family where Index column denotes the index of the incorrect error-handler, Strength denotes the testing strength that is satisfied by the family of test suites and FDE denotes the computed FDE value.

According to the $t$-factor fault model, a testing strength of $t = 2$ is required to guarantee that the incorrect error-handler with index $= 1$ is detected. However, the fault is also triggered by $68\%$ of all test suites in the test suite family that only satisfies $t = 1$.

Higher testing strengths become even more effective. As expected, a family of test suites that only satisfies $t = 3$ always detects the incorrect error-handler at index $= 2$. However, incorrect error-handlers at indices 3 and 4 are always detected as well. The incorrect error-handlers at index 5 are detected in $94\%$ of all test suites.

When increasing the number of parameters, the FDE hardly changes for incorrect error-handlers with indices between 0 and 5. The average difference between the best and worst FDE for 6, 12 and 18 parameters is $0.028\%$.

When considering higher indices of incorrect error-handlers, the required testing strength to detect a fault grows more slowly. For instance, $t = 4$ is sufficient to detect an incorrect error-handler at index 7 in $100\%$ of all cases and $t = 5$ is sufficient to detect a fault for index 10 in $100\%$ of all cases and even a fault for index 11 is detected in $98\%$ of all cases. For test scenarios with 18 parameters, $t = 5$ is sufficient to test up to an index of 12. Afterwards, the FDE decreases to $0.46\%$ for index 16 and to $0.32\%$ for index 17.

If the number of valid values is increased from 2 to 5 while not changing the number of parameters, the FDE values increase as well. For instance, $t = 2$ is sufficient to detect an incorrect error-handler up to index 5 for the $6-5-1$ family of test scenarios. For the $18-5-1$ family of test scenarios, $t = 4$ is almost sufficient for all indices of incorrect error-handlers. The FDE value for index 16 is $100\%$ and the one for index 17 is $98\%$.

This result is also consistent with the characteristics of valid values when applying the $t$-factor fault model. The FDE is improved because additional MFICs are created by the additional valid values.

If in contrast the number of invalid values increases from 1 to 4, the FDE decreases. For instance, the FDE of $t = 3$ for
index 3 decreases from 94% for 6-2-1 to 64% for 6-2-4.

The overall worst FDE is computed for 6-2-4 family of test scenarios. For index 4, the fault was not detected once by the test suite family that satisfies only $t = 1$.

For 12 parameters, a testing strength of $t = 5$ is only sufficient to detect an incorrect error-handler up to index 8. For index 9, the FDE decreases to 68% and for index 11 or higher, the FDE decreases to 8% or lower.

This finding is also consistent with the characteristics of invalid values when applying the $t$-factor fault model. The parameter of one invalid value belongs to the infection sub-condition and improves the probability of triggering the fault. But, all other invalid values either deteriorate the probability because their respective parameter belongs to the effective prevention sub-condition or the invalid value has no effects.

C. Average Fault Detection Effectiveness

Table II lists the AFDE values for all test suite families and the testing strengths from $t = 1$ to $t = 5$. Similar to FDE, adding valid values to test scenarios has a positive effect and adding invalid values has a negative effect.

Adding additional parameters has the highest impact on changing the AFDE. As discussed in the prior subsection, additional parameters have almost no impact on the FDE of existing parameters. But, the FDE for the additional parameters is worse because more parameters belong to the prevention sub-condition. Since AFDE represents the average FDE among all indices of incorrect error-handlers, the worse FDE of additional parameters decreases the average.

The AFDE values allow to draw conclusions regarding the effectiveness of CT assuming that one error-handler is incorrect when only the number of checked parameters as well as the number of valid and invalid values is known.

From that perspective, even a testing strength of $t = 2$ is sufficient to detect an incorrect error-handler in two favourable test scenario families (6-5-1 and 12-5-1) with a probability of 96% or 100%. A testing strength of $t = 3$ is sufficient to detect an incorrect error-handler in four out of nine test scenario families with a probability of 90% or higher. A testing strength of $t = 4$ has a probability of 90% or higher in six out of nine scenario families.

The effectiveness of CT is high for test scenario families with few parameters that are involved in error-handling, for test scenario families with many valid values and for test scenario families with few invalid values. In contrast, the effectiveness of CT is low for test scenario families with many parameters that are involved in error-handling and for test scenarios families with many invalid values. In the worst case, a testing strength of $t = 5$ has a probability of only 57% to detect an incorrect error-handler in 18-2-4.

To summarize the findings, CT triggers all $t$-factor faults as guaranteed by the $t$-wise coverage criterion. Furthermore, even more faults are triggered via collateral coverage. Test suites that satisfy higher testing strengths $t \geq 3$ trigger many incorrect error-handlers with higher indices.

In general, adding valid values to the test suite increases FDE and AFDE. Conversely, adding invalid values decreases FDE and AFDE. Additional parameters with error-handling have no impact on FDE for existing error-handlers. But, the faults in the additional error-handlers are hard to detect. Therefore, AEFD deteriorates with an increasing number of parameters involved in error-handling.

VIII. Threats to Validity

We compared the effectiveness of CT in the presence of error-handling and invalid values. Therefore, test inputs are generated and executed on test scenarios. Publicly available test suites are used to avoid bias in the test input generation.

The used test scenarios are artificial and do not necessarily represent realistic scenarios. In addition, it is possible that we unconsciously designed the test scenarios in a way that their pre-established characteristics are supported. However, the considered characteristics are explicit and all information is available online so that it is comprehensible and repeatable.

To prevent falsified results due to accidental fault triggering, the parameters and values of each test suite are randomized and 50 variants of each test suite are combined to a test suite family. All presented numbers are average numbers.

IX. Conclusion

CT is a generally effective approach to black-box test input generation. When invalid values are included to also test for robustness, the input masking effect can prevent faults from being triggered.

CRT is an extension to CT that incorporates robustness testing. It avoids the input masking effect by generating separate test suites for valid and invalid test inputs. But, additional information is required to separate valid from invalid inputs.

So far, the implications of input masking on the effectiveness of CT are unclear beyond the general idea. Therefore, it is unclear when to use CT and when to use CRT and accept the additional effort.

In this paper, we designed and conducted a controlled experiment to measure the effectiveness of CT in different test
scenarios and to derive concrete recommendations that can be applied in real world situations.

As a first step, we applied the t-factor fault model and discussed characteristics that are specific to error-handling and invalid values. Based on these characteristics, artificial test scenarios are designed and FDE and AFDE of test suites are measured.

The results of the experiment show that CT triggers all t-factor faults as guaranteed by the t-wise coverage criterion. Even more faults are triggered via collateral coverage. Test suites that satisfy higher testing strengths \( t \geq 3 \) trigger many incorrect error-handlers with higher indices.

Valid values increase FDE and AFDE and invalid values decrease them. Additional parameters with error-handling have the highest impact which deteriorates AFDE the most. While \( t = 2 \) is sufficient in favourable cases with few parameters involved in error-handling and with many valid values, not even \( t = 5 \) is sufficient in unfavourable cases with many parameters and with many invalid values.

In future work, we will extend the experiment to also include invalid value combinations. Further, we will compare not only the effectiveness but also the efficiency of CT with CRT.

REFERENCES


