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, input masking 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**

CEUR-WS.org/Vol-2511/QuASoQ-04.pdf

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 the faults are 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 generated 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]. It is argued that CRT is necessary because of the *input masking effect* [5], [9]–[11]: The first invalid value that is evaluated by the SUT triggers error-handling and the normal control-flow is left. 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. Therefore, the test model must be enriched with additional semantic information about invalid values. Previous experiments [5] have shown that CRT is an effective approach in the presence of error-handling.

Despite the presence of error-handling, CT can also reveal failures without requiring additional semantic information. However, the extent to which the input masking effect can impact the fault detection effectiveness is is yet unknown. 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 use a customer registration service as an example. The example consists of 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.

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

$$\begin{array}{ll} p_1 : Title & V_1 = \{Mr, Mrs, 123\} \\ p_2 : FamilyName & V_2 = \{Miller, Davis, 123\} \\ p_3 : Address & V_3 = \{UK, US, 123\} \end{array}$$

Figure 1. Example Test Model

Further, assume that the implementation of the service contains a fault in the validity check for family names that returns a wrong error code (title instead of name error). It is triggered whenever an invalid family name, e.g. `[FamilyName:123]`, is evaluated. An implementation is illustrated in Listing 1 with `INV_xxx` string literals representing specific error codes.

```
String register(String title, family, addr){
 if(isInvTitle(title)) return INV_TITLE;
 if(isInvFamilyName(family)) return INV_TITLE;
 if(isInvAddress(addr)) return INV_ADDRESS;
 ...
}
```

Listing 1. Example of an Input Validity Check

A test input like `[Title:Mrs, FamilyName:123, Address:UK]` would trigger the fault. In contrast, a test input like `[Title:123, FamilyName: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 `[FamilyName:123]`. In total, nine test inputs with `[FamilyName: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 $\frac{6}{9} = 66\%$.

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 [12]. 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 $P = \{p_1, ..., p_n\}$ and the domain of each parameter p_i is a finite nonempty set of m_i values $V_i = \{v_1, ..., v_{m_i}\}$. A **combination** τ 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 τ_a covers another combination τ_b if every parameter-value pair of τ_b is included in τ_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 and a **combination strategy** describes how to select test inputs T 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 [13]. This so-called **collateral coverage** can potentially help triggering additional faults [14].

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 unsatisfied.

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 τ 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 τ 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. [9] 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. [11] discussed an example where input masking prevented a fault from being triggered. A case study by Wojciak and Tzoref-Brill [15] reported on CT including testing with invalid inputs.

The CT tools AETG [9], ACTS [16] and PICT [10] allow to mark individual values as invalid and also generate separate

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

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 that compared CRT with CT. But, they focused on configuration-dependent faults where the error-handling depends on a certain configuration of valid parameter combinations. Further, 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.

Other empirical studies also compared the efficiency of CT. A recent study summarizes previous comparisons [21]. However, none of these studies focused on error-handling.

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 $p_1, ..., p_n$ with each parameter p_i having a domain D_i consisting of a potentially infinite number of values.

The faults are 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 τ that triggers a t-factor fault contains a combination c that is a **failure-inducing combination** (FIC). Each test input τ that covers $c \subseteq \tau$ triggers a fault [31]. A FIC c is **minimal** (MFIC) if no proper subset $c' \subset c$ 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 family 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 `isInvFamilyName(family)` triggers the fault. Consequently, `[FamilyName:123]` is not a FIC. Triggering the fault requires a valid title because the error-handling `isInvTitle(title)` propagates otherwise. Therefore, the fault can be modelled as a 2-factor fault using a conjunction over `title` and `family`: $\neg$(`isInvTitle(title)`) $\wedge$ (`isInvFamilyName(family)`).

For the given test model, the combinations `[Title:Mr, FamilyName:123]` and `[Title:Mrs, FamilyName: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 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, $\neg$(`isInvTitle(title)`) is used for prevention and (`isInvFamilyName(family)`) 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 termi-

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

nate 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)` $\vee$ `isInvFamilyName(family)`).

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 family name becomes a 3-factor fault.

To make the determination of testing strength t independent from the location of an 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 `FamilyName` are involved in the condition $\neg$`(isInvTitle(title))` $\wedge$ `(isInvFamilyName(family))` 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 $\frac{2}{3} = 66\%$ to trigger the fault, i.e. at least one test input covers `[FamilyName:123]` and there is a $\frac{2}{3}$ 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 family name `[FamilyName:Smith]` to the example test model does not affect the FICs because it cannot satisfy `(isInvFamilyName(family))`.

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, FamilyName:123]`. For 1-wise testing, the probability to trigger the fault increases to $\frac{3}{4} = 75\%$.

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 `[FamilyName: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 $\frac{6}{9} = 66\%$. When another valid address is added, 12 test inputs that cover `[FamilyName: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 may affect the overall selection of test inputs. Maybe a combination strategy cannot find a small test suite 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. 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 $\frac{2}{4} = 50\%$.

When the parameter of the invalid value is involved

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

in the infection sub-condition, the probability of input masking is decreased. For instance, adding another invalid value `[FamilyName:456]` that triggers the same fault as `[FamilyName:123]` adds two more MFICs `[Title:Mr, FamilyName:456]` and `[Title:Mrs, FamilyName: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 source code and the results of the experiment are available at our companion website[1].

For an experiment, it is important to control the factors that can influence the results. Therefore, we designed artificial test scenarios and changed different characteristics in a controlled and traceable way. Each test scenario contains exactly one faulty error-handler which always propagates to a failure when triggered and the failure is always revealed by the test oracle. Thus, the selection of test inputs is the only factor that influences the results of test execution.

The implementation of a test scenario is illustrated in Listing 2. A test scenario accepts input values for a number of parameters and includes a sequence of error-handlers with one error-handler for one parameter implemented by an `if` statements. The result of a test scenario execution is `INV_INPUT` if an error-handler correctly identifies an invalid value and terminates the execution. If an invalid value is identified by a faulty implemented error-handler, `NULL` is returned instead; `VAL_INPUT` is returned if all error-handlers are passed because all values are valid.

```
String checkInput (Object a, b, c){
  if(isInvalid(a)) return INV_INPUT;
  else if(isInvalid(b)) return NULL;
  else if(isInvalid(c)) return INV_INPUT;
  else return VAL_INPUT;
}
```

Listing 2. Illustration of a Test Scenario Implementation

[1] https://github.com/coffee4j/quasoq-2019

Table I
Test Suite Sizes used for the Experiment

P	V	t=1	t=2	t=3	t=4	t=5
6	2	2	6	14	26	42
6	3	3	15	49	140	318
6	4	4	24	109	442	1377
6	5	5	38	209	1059	4195
6	6	6	52	361	2165	10407
12	2	2	8	19	47	105
12	3	3	20	71	262	885
12	4	4	31	170	835	3854
12	5	5	48	329	2068	11889
12	6	6	67	564	4295	29645
18	2	2	10	24	60	144
18	3	3	20	88	336	1241
18	4	4	36	202	1098	5458
18	5	5	54	401	2725	16843
18	6	6	75	689	5650	42102
24	2	2	10	26	71	175
24	3	3	21	98	396	1513
24	4	4	40	232	1298	6629
24	5	5	59	454	3203	20482
24	6	6	83	786	6662	51287
30	2	2	11	28	80	200
30	3	3	23	106	446	1715
30	4	4	41	255	1448	7555
30	5	5	63	495	3589	23369
30	6	6	86	859	7473	58468

Based on the application of the t-factor fault model, we describe each test scenario S in terms of (1) the number of parameters, (2) the number of parameters that are involved in error-handling, (3) the number of valid values, (4) the number of invalid values per parameter, and (5) the index of the position of the incorrect error-handler that contains a fault.

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-E` where `P` refers to the number of parameters involved in error-handling, `V` refers to the number of valid values per parameter, `I` refers to the number of invalid values per parameter, and `E` refers to the number of parameters that are involved in error-handling.

The experiment starts with a *root* test scenario family `6-1-1-6` representing a simple application of CT. The root test scenario family is then extended as follows. The number of parameters is increased by six up to 30 parameters. The number of error-handlers in a test scenario family either remains at six or is equal to the number of parameters. The total number of number of values per test scenario family is extended up to six with one to five valid and invalid values.

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

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

valid and invalid values per parameter. To avoid any bias, we use test suites from the NIST Covering Array Tables[2]. They are publicly available and contain many of the smallest known test suites [32]. Table I depicts the sizes of the test suites. Column *P* refers to the number of parameters and column *V* refers to the total number of values per parameter. The test suites are reused for different ratios of valid and invalid values.

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 100 different variants of each test 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) [8], [14].

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

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

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.

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

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.

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

[2] https://math.nist.gov/coveringarrays/

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 1-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 $(1 \leq t \leq 5)$, all $(\text{index} + 1)$-factor faults are triggered by all test suites that satisfy the corresponding testing strength.

Beyond that, collateral coverage causes higher $(\text{index} + 1)$-factor faults to be repeatedly triggered by test suites that satisfy lower testing strengths.

B. Fault Detection Effectiveness

Table II depicts an excerpt of FDEs computed for test scenario families with six to 30 parameters consisting of two valid values and one invalid value. Each test scenario family contains six error-handlers. The *I* column denotes the index of the incorrect error-handler, t denotes the testing strength that is satisfied by the family of test suites and the remaining columns denote the computed FDE values.

According to the t-factor fault model, a testing strength of $t = 2$ is required to guarantee that an incorrect error-handler with index = 1 is detected. Among all depicted test scenario families, the fault is on average also triggered by 67.6% test suite families that only satisfy $t = 1$. The exact numbers are depicted in the second row of Table II.

Testing with higher testing strengths is even more effective. On average, test suite families that satisfy a testing strength of $t = 2$ detects incorrect error-handlers at index = 2 in 99.2% of all cases. As expected, a family of test suites that satisfies $t = 3$ always detects the incorrect error-handlers 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 by 98.6% of all test suite families.

When increasing the number of parameters while the number of error-handlers remains at six, the data indicates no general trend for the FDE. In many cases, the FDE improves slightly. Although, there are cases where the FDE deteriorates. For instance, the FDE for detecting an incorrect error-handler at index 1 with a test suite family that satisfies only testing strength $t = 1$ improves from 64% for the test scenario family with six parameters to 74% for the test scenario family with 12 parameters. But, it deteriorates to 65% for the test scenario family with 18 parameters.

Table III depicts the FDE for test scenario families with six to 30 parameters and an equal number of error-handlers. Increasing the number of error-handlers such that one error-handler exists for each parameter has no direct impact on the

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Table II
FDE FOR TEST SCENARIO FAMILIES WITH SIX TO 30 PARAMETERS AND SIX ERROR-HANDLERS (EXCERPT)

I	t	6-2-1-6	12-2-1-6	18-2-1-6	24-2-1-6	30-2-1-6
0	1	100	100	100	100	100
1	1	63	74	65	66	70
1	2	100	100	100	100	100
2	1	43	34	38	35	52
2	2	98	99	100	99	100
2	3	100	100	100	100	100
3	1	30	35	33	29	31
3	2	90	96	95	95	99
3	3	100	100	100	100	100
4	1	22	22	20	19	27
4	2	72	82	84	85	91
4	3	100	100	100	100	100
5	1	12	14	11	13	18
5	2	61	61	69	63	69
5	3	95	99	100	100	99
5	4	100	100	100	100	100

Table III
FDE FOR TEST SCENARIO FAMILIES WITH SIX TO 30 PARAMETERS AND AN EQUAL NUMBER OF ERROR-HANDLERS (EXCERPT)

I	t	6-2-1-6	12-2-1-12	18-2-1-18	24-2-1-24	30-2-1-30
0	1	100	100	100	100	100
1	1	63	63	66	77	66
1	2	100	100	100	100	100
2	1	43	48	55	39	49
...	...	...	...	...	...	...
7	3		78	86	91	87
9	3		52	52	61	55
9	4		95	96	98	99
11	3		34	29	30	34
11	4		67	78	88	83
11	5		99	100	100	100
17	3			2	2	2
17	4			13	16	16
17	5			38	36	44
23	4				2	2
23	5				3	5

FDE. Over all indices, testing strengths and parameter sizes, the difference between the FDE for six error-handlers and the FDE for $|P|$ error-handlers is 1.25 percentage points on average. For instance, the difference for $I = 1$, $t = 1$ and 12 parameters is 11 percentage points with an FDE of 74% for 6 error-handlers (Table II) and an FDE of 63% for 12 error-handlers (Table III).

The biggest deviations are noticed for test scenario families with 30 parameters. For $I = 1$ and $t = 1$, the difference between the FDE for six error-handlers (57%) and the FDE for 30 error-handlers (42%) is 15 percentage points. For $I = 5$ and $t = 2$, the difference between the FDE for six error-handlers (40%) and the FDE for 30 error-handlers (60%) is -20 percentage points.

Analysing higher indices of incorrect error-handlers ($I > 5$) emphasizes that the required testing strength to detect a fault increases as well. Although, the testing strength grows slower compared to lower indices ($I \leq 5$). For instance, $t = 3$ is sufficient to detect incorrect error-handlers at index $I = 4$ in 100% of all cases and it is almost sufficient (98.6% on average) to detect all incorrect error-handlers at index $I = 5$. For $I = 7$, the average FDE decreases to 85.5%. For $I = 17$, only 2% of all incorrect error-handlers are detected. In comparison, $t = 4$ is sufficient to detect an incorrect error-handler at index $I = 7$ in 100% of all cases. $t = 5$ is sufficient to detect a fault for index $I = 10$ in 100% of all cases and even a fault for index $I = 11$ is detected in 99.75% of all cases. Afterwards, the FDE for $t = 5$ decreases to an average of 39.3% for index $I = 17$ and to 4% for $I = 23$.

Overall, the findings are consistent with the application of the t-factor fault model. Changing the number of parameters of a test scenario family has no clear effect on the FDE as well as changing the total number of error-handlers. In contrast, increasing or decreasing the specific index of the incorrect error-handler has the biggest effect on the FDE.

Following the t-factor fault model, a testing strength t guarantees to detect incorrect error-handlers up to index $I = t - 1$ because one parameter is involved in the infection sub-condition. Beyond that, the collateral coverage effect of CT causes a reliable detection of incorrect error-handlers with higher indices. Although for index $I = 11$ and higher, even $t = 5$ is not sufficient to detect incorrect error-handlers for the test scenario families depicted in Table III.

To use the knowledge acquired from FDE metric, knowledge about the index of an incorrect error-handler is required which makes it hard to use the information in practice. In contrast, 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. Therefore, all further analyzes are based on the AFDE metric.

C. Average Fault Detection Effectiveness

Table IV and Table V list the AFDE values for all test suite families and the testing strengths from $t = 1$ to $t = 5$. Column P denotes the number of parameters, column E denotes the number of error-handlers, and column t denotes the testing strength. *Avg.* depicts the average AFDE among all testing strengths. The remaining columns follow the pattern `V-I` with `V` describing the number of valid values and `I` describing the number of invalid values. Table IV contains the AFDE values for test scenario families with six error-handlers. In Table V, the number of error-handlers equals the number of parameters.

As discussed in the prior subsection, increasing the number of parameters only has no clear effect on the FDE. The AFDE metric reflects this as depicted by Table IV.

Increasing the total number of error-handlers had no impact on the FDE of existing error-handling indices. But, the FDE for the additional indices 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 error-handlers decreases the AFDE value. This is depicted in Table V.

Changing the number of values per parameter has a great effect on the AFDE. Depending on whether the additional values are valid or invalid, the AFDE increases or decreases.

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Table IV
AFDE OF ALL TEST SCENARIO FAMILIES WITH SIX ERROR-HANDLERS

P	E	t	1-1	1-2	1-3	1-4	1-5	2-2	3-3	2-1	3-1	4-1	5-1
6	6	1	36	33.17	34.83	34.17	33.83	47.5	53.33	45	53	63.33	65.83
6	6	2	55.33	53.83	53	53.33	52.33	79.5	94.67	86.83	97	99.5	99.83
6	6	3	71.83	71.5	73.33	70.83	70.67	98	99.83	99.17	100	100	100
6	6	4	86.67	86.67	86.5	85.33	86.33	100	100	100	100	100	100
6	6	5	95	96	96.33	97.33	98.33	100	100	100	100	100	100
6	6	*Avg.*	68.97	68.23	68.8	68.2	68.3	85	89.57	86.2	90	92.57	93.13
12	6	1	33.33	32.17	31.5	33	34	46.5	57.5	46.5	55.83	64	67.5
12	6	2	60	60.67	57.33	54.67	56	84.5	92.83	89.67	99	99.67	100
12	6	3	76.67	74.83	74.33	75	71.33	99.33	100	99.83	100	100	100
12	6	4	94.17	92.5	90.5	91	88.83	100	100	100	100	100	100
12	6	5	97.83	99.33	99.83	99.33	100	100	100	100	100	100	100
12	6	*Avg.*	72.4	71.9	70.7	70.6	70.03	86.07	90.07	87.2	90.97	92.73	93.5
18	6	1	33.33	32.83	33.33	32.83	34	43.83	52.17	44.5	57	58.33	65.83
18	6	2	62.33	58.83	57	56.67	53.5	86	94.5	91.33	97.5	100	100
18	6	3	80.83	79.67	77.33	74	73.83	99.5	100	100	100	100	100
18	6	4	94	94.17	94.17	92.17	90.5	100	100	100	100	100	100
18	6	5	99.67	100	100	100	100	100	100	100	100	100	100
18	6	*Avg.*	74.03	73.1	72.37	71.13	70.37	85.87	89.33	87.17	90.9	91.67	93.17
24	6	1	31.33	33.33	33	33.5	32.67	46.17	52.33	43.67	53.67	60.33	66.5
24	6	2	63.17	59.83	58.17	57.33	55.5	85.83	94.17	90.33	98.33	100	100
24	6	3	80.67	77.67	77.5	75.67	75	99.67	100	100	100	100	100
24	6	4	94.83	94.67	94.33	91.5	91.83	100	100	100	100	100	100
24	6	5	100	99.83	100	100	100	100	100	100	100	100	100
24	6	*Avg.*	74	73.07	72.6	71.6	71	86.33	89.3	86.8	90.4	92.07	93.3
30	6	1	35.33	34.5	33.17	33.83	32.33	45.83	51.67	49.67	54.17	60.83	69.17
30	6	2	65.17	61.67	57.17	57.17	56.67	84.83	95	93.17	98.83	100	100
30	6	3	84.33	83.5	79	77	72.5	100	100	99.83	100	100	100
30	6	4	95.83	96.67	93.5	91.33	93	100	100	100	100	100	100
30	6	5	99.83	99.83	100	100	100	100	100	100	100	100	100
30	6	*Avg.*	76.1	75.23	72.57	71.87	70.9	86.13	89.33	88.53	90.6	92.17	93.83

Table V
AFDE OF ALL TEST SCENARIO FAMILIES WITH INCREASING NO. OF ERROR-HANDLERS

P	E	t	1-1	1-2	1-3	1-4	1-5	2-2	3-3	2-1	3-1	4-1	5-1
6	6	1	36	33.17	34.83	34.17	33.83	47.5	53.33	45	53	63.33	65.83
6	6	2	55.33	53.83	53	53.33	52.33	79.5	94.67	86.83	97	99.5	99.83
6	6	3	71.83	71.5	73.33	70.83	70.67	98	99.83	99.17	100	100	100
6	6	4	86.67	86.67	86.5	85.33	86.33	100	100	100	100	100	100
6	6	5	95	96	96.33	97.33	98.33	100	100	100	100	100	100
6	6	*Avg.*	68.97	68.23	68.8	68.2	68.3	85	89.57	86.2	90	92.57	93.13
12	12	1	17	16	16.58	16.5	16.67	23.08	26.33	25.42	32.08	38.75	44.5
12	12	2	29.5	30	27.83	26.83	26.92	46.58	54.17	54.08	74.75	89.58	95.83
12	12	3	41.92	39.5	38.08	36.75	36.08	64.92	78.33	79.67	98.25	100	100
12	12	4	52	49.5	46.83	45.67	45.17	81.42	96.75	95.58	100	100	100
12	12	5	61.33	57.83	55.33	55.08	54.75	94.67	100	99.92	100	100	100
12	12	*Avg.*	40.35	38.57	36.93	36.17	35.92	62.13	71.12	70.93	81.02	85.67	88.07
18	18	1	11	11.17	11.28	11	10.72	15.78	18.61	17.11	23.22	27.94	32.78
18	18	2	22.33	19.72	19.28	18.5	17.5	31.83	36.67	39.22	58.94	72.67	84.72
18	18	3	28.61	27.72	26	24.83	24.22	44.78	54.78	58.11	84.56	98.78	99.83
18	18	4	36.39	33.83	32.44	31	30.83	58.56	70.94	74.33	98.5	100	100
18	18	5	42.67	40.11	38.72	37.56	36.56	70.28	86.5	90.17	100	100	100
18	18	*Avg.*	28.2	26.51	25.54	24.58	23.97	44.24	53.5	55.79	73.04	79.88	83.47
24	24	1	8.25	8.38	8.46	8.04	8.63	11.79	13.38	11.75	16.71	21.08	32.06
24	24	2	16.46	14.96	14.08	14.58	13.75	23.29	29.33	31.04	47.88	63.25	87.5
24	24	3	22.08	19.67	19.88	19.29	19.13	34.46	41.33	49	80.75	98.08	99.94
24	24	4	27.83	26.54	24.71	24.38	23.67	45.04	54.71	72.79	99.33	100	100
24	24	5	33.92	31.21	29.21	28.79	28.58	54.75	69.29	92.58	100	100	100
24	24	*Avg.*	21.71	20.15	19.27	19.02	18.73	33.72	40.79	43.37	64.3	73.78	79.32
30	30	1	6.3	6.93	6.3	6.47	6.83	9.07	10.8	10.3	13.5	17	19.73
30	30	2	13.57	12.1	11.8	11.47	10.93	19.53	22.53	23.57	35.73	48.87	60.5
30	30	3	18.03	16.67	15.6	15.73	15.07	27.8	34.17	36.17	57.87	77.53	92.93
30	30	4	23.1	20.77	19.77	19.3	19.03	35.77	44	47.47	75.2	96.6	100
30	30	5	27.13	24.97	23.67	23.07	22.8	44.3	53.7	58.57	91.87	100	100
30	30	*Avg.*	17.63	16.29	15.43	15.21	14.93	27.29	33.04	35.21	54.83	68	74.63

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

The AFDE always improves when adding only valid values. For the largest test scenario $P = 30$ and $E = 30$ with four and five valid values, the testing strengths $t = 4$ and $t = 5$ are sufficient to detect an incorrect error-handler almost always. Although, testing with $t = 5$ can be perceived as impractical as it would require the execution of 23369 (`4-1`) and 58468 (`5-1`) test cases (See Table I).

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.

Besides the two favorable cases with (`4-1`) and (`5-1`), $t = 5$ is not sufficient to detect all incorrect error-handlers reliably. For 2 valid values (`2-1`) and $E = 18$, the AFDE is 90.17%. A higher testing strength would be necessary which further increases the time for test input generation and the time for test execution due to the larger the test suite size.

There is also a trend indicating that the AFDE decreases when adding only invalid values. For instance, the average AFDE for $P = 30$ and $E = 6$ decreases from 76.1% for one valid and one invalid value (`1-1`) to 70.9% for five invalid values (`1-5`). However, this trend is not as clear as for valid values. One exception exists for $P = 6$ and $E = 6$ where the average AFDE improves by 0.57 (`1-3`) and 0.1 (`1-5`) percentage points.

But, the trend 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 effect.

When adding valid and invalid values equally (`2-2` and `2-2`), the AFDE always increases in comparison to `1-1`. The AFDE is also always higher when comparing it to test scenario families with more invalid than valid values. But, the AFDE is always lower compared to test scenario families with more valid than invalid values.

This finding is still consistent with the t-factor fault model. Although, it cannot be directly derived from it. The numbers indicate that the additional MFICs introduced by additional valid values have a stronger effect on the AFDE than the prevention sub-conditions increased by additional invalid values.

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.

Adding valid values to the test suite increases the AFDE. But, the required test suite size increases as well. Conversely, adding invalid values decreases the AFDE. Additional parameters with error-handling have no impact on FDE for existing error-handlers. But, the faults in the additional error-handlers are harder to detect. Therefore, AFDE deteriorates with an increasing number of parameters involved in error-handling.

Overall, the experiment shows that CT can be an effective approach to detect incorrect error-handlers. Although, not all incorrect error-handlers can be detected reliably. Depending on the number of error-handlers and the distribution of valid and invalid values, a high testing strength is necessary which requires the execution of large test suites.

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[1] so that it is comprehensible and repeatable. In addition, the AFDE metric is designed to derive knowledge and to apply it to real-world scenarios.

To prevent falsified results due to accidental fault triggering, the parameters and values of each test suite are randomized and 100 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 considering invalid values to also test for robustness, the input masking effect can prevent faults from being triggered. CRT is an extension to CT for robustness testing that avoids the input masking effect. But, CRT imposes extra costs since it requires additional semantic information in the test model. The implications of input masking on the effectiveness of CT are unclear beyond the general idea. Therefore, it is unclear when CT can be used and when CRT should be used despite the extra costs.

In this paper, we designed and conducted a controlled experiment to measure the effectiveness of CT in different test scenarios. Therefore, 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 tested using publicly available small test suites.

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 $t = 4$ is sufficient in favourable cases with many valid values and few parameters involved in error-handling, not even $t = 5$ is sufficient in unfavourable cases with many invalid values and many parameters involved in error-handling.

Overall, the experiment shows that CT can be effective in the presence of error-handling and invalid values. But in

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

many cases, an advantageous distribution of valid and invalid values, a high testing strength and thus very large test suites are required.

Despite the good performance of CT in many cases, CRT is a promising approach that requires potentially fewer test executions. Although, a direct comparison is necessary to decide if the execution of very large test suites required by CT outweighs the additional modelling effort.

In future work, we will extend the experiment to consider invalid value combinations and configuration-dependent faults as well. As they potentially increase the number of parameters involved in the prevention and infection sub-conditions. Further, we will compare not only the effectiveness but also the efficiency of CT with CRT.

References

[1] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std*, vol. 610.12-1990, 1990.

[2] A. Avizienis, J. Laprie, B. Randell, and C. E. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Sec. Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[3] M. Young and M. Pezze, *Software Testing and Analysis: Process, Principles and Techniques*. USA: John Wiley & Sons, Inc., 2005.

[4] Y. Li, S. Ying, X. Jia, Y. Xu, L. Zhao, G. Cheng, B. Wang, and J. Xuan, "Eh-recommender: Recommending exception handling strategies based on program context," in *23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018, Melbourne, Australia, December 12-14, 2018*, 2018, pp. 104–114.

[5] K. Fögen and H. Lichter, "Combinatorial robustness testing with negative test cases," in *2019 IEEE International Conference on Software Quality, Reliability and Security, QRS 2019, Sofia, Bulgaria, July 22-26, 2019*, 2019, pp. 34–45.

[6] P. Sawadpong, E. B. Allen, and B. J. Williams, "Exception handling defects: An empirical study," in *14th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2012, Omaha, NE, USA, October 25-27, 2012*, 2012, pp. 90–97.

[7] N. Li and J. Offutt, "Test oracle strategies for model-based testing," *IEEE Trans. Software Eng.*, vol. 43, no. 4, pp. 372–395, 2017.

[8] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: a survey," *Softw. Test., Verif. Reliab.*, vol. 15, no. 3, pp. 167–199, 2005.

[9] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatiorial design," *IEEE Trans. Software Eng.*, vol. 23, no. 7, pp. 437–444, 1997.

[10] J. Czerwonka, "Pairwise testing in real world: Practical extensions to test case generators," in *24th Pacific Northwest Software Quality Conference*, vol. 200, 2006.

[11] M. Grindal, B. Lindström, A. Offutta, and S. Andler, "An evaluation of combination strategies for test case selection," Department of Computer Science, University of Skövde, Tech. Rep. HS-IDA-TR-03-001, 2003.

[12] Y. Lei and K. Tai, "In-parameter-order: A test generation strategy for pairwise testing," in *3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE '98), 13-14 November 1998, Washington, D.C, USA, Proceedings*, 1998, pp. 254–261.

[13] B. Chen and J. Zhang, "Tuple density: a new metric for combinatorial test suites," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, 2011, pp. 876–879.

[14] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection," *IEEE Trans. Software Eng.*, vol. 41, no. 9, pp. 901–924, 2015.

[15] P. Wojciak and 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.

[16] L. Yu, Y. Lei, R. Kacker, and D. R. Kuhn, "ACTS: A combinatorial test generation tool," in *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013*, 2013, pp. 370–375.

[17] K. Fögen and H. Lichter, "Combinatorial testing with constraints for negative test cases," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops, Västerås, Sweden, April 9-13, 2018*, 2018, pp. 328–331.

[18] ——, "Repairing over-constrained models for combinatorial robustness testing," in *2019 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2019, Sofia, Bulgaria, July 22-26, 2019*, 2019, pp. 177–184.

[19] ——, "Semi-automatic repair of over-constrained models for combinatorial robustness testing," in *26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, Dec 2-5, 2019 (to be published)*, 2019.

[20] ——, "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.* CEUR-WS.org, 2018, pp. 22–29.

[21] H. Wu, N. Changhai, J. Petke, Y. Jia, and M. Harman, "An empirical comparison of combinatorial testing, random testing and adaptive random testing," *IEEE Transactions on Software Engineering*, 2018.

[22] S. R. Dalal and C. L. Mallows, "Factor-covering designs for testing software," *Technometrics*, vol. 40, no. 3, pp. 234–243, 1998.

[23] D. Harel and A. Pnueli, "On the development of reactive systems," in *Logics and Models of Concurrent Systems*, K. R. Apt, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 477–498.

[24] D. R. WALLACE and D. R. KUHN, "Failure modes in medical device software: An analysis of 15 years of recall data," *International Journal of Reliability, Quality and Safety Engineering*, vol. 08, no. 04, pp. 351–371, 2001.

[25] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings.*, Dec 2002, pp. 91–95.

[26] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Trans. Software Eng.*, vol. 30, no. 6, pp. 418–421, 2004.

[27] K. Z. Bell and M. A. Vouk, "On effectiveness of pairwise methodology for testing network-centric software," in *2005 International Conference on Information and Communication Technology*, Dec 2005, pp. 221–235.

[28] D. R. Kuhn and V. Okun, "Pseudo-exhaustive testing for software," in *30th Annual IEEE / NASA Software Engineering Workshop (SEW-30 2006), 25-28 April 2006, Loyola College Graduate Center, Columbia, MD, USA*. IEEE Computer Society, 2006, pp. 153–158.

[29] D. R. Kuhn, R. N. Kacker, and Y. Lei, "Estimating t-way fault profile evolution during testing," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, June 2016, pp. 596–597.

[30] R. N. Kacker, D. R. Kuhn, Y. Lei, and J. F. Lawrence, "Combinatorial testing for software: An adaptation of design of experiments," *Measurement*, vol. 46, no. 9, pp. 3745 – 3752, 2013.

[31] P. Arcaini, A. Gargantini, and M. Radavelli, "Efficient and guaranteed detection of t-way failure-inducing combinations," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2019, Xi'an, China, April 22-23, 2019*, 2019, pp. 200–209.

[32] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and R. D. Kuhn, "Refining the in-parameter-order strategy for constructing covering arrays," *Journal of Research of the National Institute of Standards and Technology*, vol. 113, no. 5, pp. 287–297, 09 2008.

Copyright © 2019 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).