Chapter 1
Effectiveness of Combinatorial Test Design with Executable Business Processes

Daniel Lübke, Joel Greenyer, and David Vatlin

Abstract Executable business processes contain complex business rules, control flow, and data transformations, which makes designing good tests difficult and, in current practice, requires extensive expert knowledge. In order to reduce the time and errors in manual test design, we investigated using automatic Combinatorial Test Design (CTD) instead. CTD is a test selection method that aims at covering all interactions of a few input parameters. For this investigation, we integrated CTD algorithms with an existing framework that combines equivalence class partitioning with automatic BPELUnit test generation. Based on several industrial cases, we evaluated the effectiveness and efficiency of test suites selected via CTD algorithms against those selected by an expert and random tests. The experiments show that CTD tests are not more efficient than tests designed by experts, but that they are a sufficiently effective automatic alternative.

Key words: Executable Business Processes, Testing, Combinatorial Test Design, Industrial Case Study, IPOG, AETG

1.1 Introduction

Many organizations rely on *executable business processes* (XBPs) to orchestrate distributed services in order to satisfy critical business needs. Therefore, the correctness of each XBP must be thoroughly validated via tests. However, complex business rules, process-flow, and data transformations make it difficult to engineer effective test suites XBPs [11].

Leibniz Universität Hannover, Fachgebiet Software Engineering,
Welfengarten 1, D-30167 Hannover, Germany
e-mail: {daniel.luebke | greenyer}@inf.uni-hannover.de

Ideally, to catch all bugs, an XBP should be tested with all possible combinations of input parameter values, but this is intractable in practice. Equivalence class partitioning can help abstract from the single input values and partition the range for each input parameter into a few sets of values where it is assumed that the XBP exhibits equivalent behavior. Sometimes, it is then possible to cover all combinations of the parameters' equivalence classes. Usually, however, this will still require too many tests; even with automated tests, a single XBP test can take up to minutes, and is thus resource-intensive.

In the industrial project Terravis, which develops a process integration platform between land registers, notaries, and banks throughout Switzerland [2], approximately hundred XBPs are constantly improved and extended, so tests must be designed, maintained, and executed regularly. For efficient systematic testing, the *classification tree generator* framework (CTG) [19] was developed, which combines equivalence class partitioning via classification trees [6] with automated generation of BPELUnit [16, 11] tests.

CTG aids testers in defining equivalence classes and selecting and generating tests. The test selection determines which input messages an XBP under test receives. Moreover, requirements can be formalized as constraints, so that also assertions on the output values can be generated automatically.

Selecting a good test set, however, is still a time-consuming, and error prone expert task. To address this problem, we investigate employing *Combinatorial Test Design* (CTD) [21] to automate the selection of effective and efficient tests. CTD is a test planning approach which relies on the observation that whether a software bug is executed, which is necessary for finding it in a test, usually depends on the interaction of only a few input parameters [20, 8]. For example, if a system has three Boolean input parameters, there are eight ($|\mathbb{B} \times \mathbb{B} \times \mathbb{B}|$) tests for the system. All pairwise combinations of these parameter values, however, would already be covered by five test cases, for example those with inputs $(0,0,0)$, $(0,1,1)$, $(1,0,0)$, $(1,0,1)$, $(1,1,0)$—and these would be sufficient for finding any bug that depends on any particular value combination of any two parameters. When dealing with many input parameters, but only covering t-wise parameter interactions where t is small, the reduction of tests w.r.t. testing all possible inputs is significant.

Automatic algorithms exist for synthesizing a sets of test inputs that cover all t-wise interactions of parameter values, for example the IPOG algorithm [10, 23]. CTD can be combined with expert-based equivalence class partitioning of input parameters as mentioned above.

Applying CTD for the testing of XBPs is currently not an established nor researched practice, so we aim to answer the following research questions:

RQ1: What value of t is typically required for CTD tests of executable business processes in order to be effective?

RQ2: How does the effectiveness and efficiency of the CTD-generated tests, with different t-values, compare to (a) tests created by experts, and (b) tests selected randomly?

In order to answer these questions, we implemented an automated CTD procedure and conducted an experiment with the industrial XBPs provided by the Terravis project. Terravis provides many large BPEL processes and is thus a good candidate for exploring the effects of testing methodologies.

Because the CTG framework and BPELUnit [16, 11] are already in use in the Terravis project, we integrated CTG with the two most common CTD algorithms, IPOG-C [10, 23] and AETG-SAT [4, 5]. Through the use of BPELUnit, we can easily measure the test case sizes and test coverage with respect to activities and decisions in the process [15]. We use the latter as a metric for the effectiveness of a test suite, i.e., its capability of discovering bugs.

To our knowledge, applying CTD to the business process domain is new, or has not been published previously. The contribution of our paper is, first, that we show CTD can be applied to XBPs. Second, we present results of experiments carried out on industrial cases; the results allows practitioners to judge when and how to apply these methods.

Structure: We introduce preliminaries in Sect. 1.2 and overview related work in Sect. 1.3. The design and results of our experiments are described in Sect. 1.4 and 1.5. Finally, we conclude in Sect. 1.6

1.2 Preliminaries

In this section, we explain the basic features of the classification tree generator framework (CTG) and how we combined it with CTD algorithms.

Suppose the process under test is a simple online shop process modeled in BPMN as shown in Fig. 1.1. Initially, a customer places an order. The product may not be available, but if it is, the customer receives an order confirmation and the freight company receives an order, upon which the freight company returns a packaging label. At this point, it may still turn out that the shipment is not possible, for example because the product is out of stock (and stock-level data was inconsistent). If the shipment is possible, it is handed to the freight company and an invoice is sent to the customer.

The order placed by the customer is a document that contains different parameters, like the ID of the ordered product, amount, payment information, customer ID, etc. Of course it is not possible to test the process for each possible order, and so we use equivalence class partitioning of the different input parameters. For example, we assume that different payment methods (credit card, Maestro, or wire transfer) will lead to different behavior, but the actual credit card number, for example, does not influence the process' behavior. Likewise, the process will behave similarly for available products for which shipment is possible, which shows that there can also be a multi-dimensional partitioning with interdependencies, for example a shipment can only be successful when the product is available.

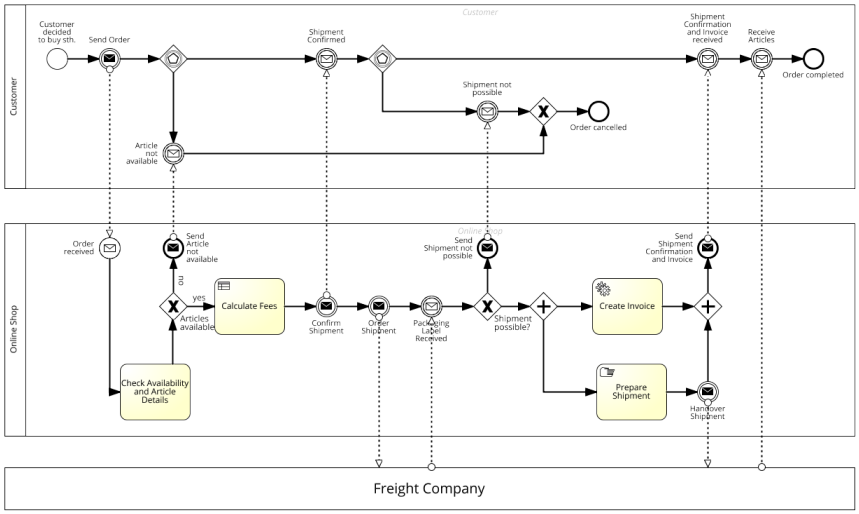

Fig. 1.1 Simple Online Shop Process

We thus use a classification-tree-based approach [6] for the selection of test inputs. A classification tree for the online shop process is shown in Fig. 1.2. It is displayed as a spreadsheet where, below the tree structure that make the table heading, tests can be configured by selecting from the equivalence classes resulting from the classification-tree-based partitioning.

	Article Available?			Shipment Size			Payment Method		
	Shipment possible?								
	Success	Problem	Not Available	Maxi Letter	Package	Bulk	Credit Card	Maestro	Wire Transfer
TestCases:									
TC1			X	X				X	
TC2			X		X				X
TC3			X			X	X		
TC4	X			X					X
TC5	X				X		X		
TC6	X					X		X	
TC7		X		X			X		
TC8		X			X			X	
TC9		X				X			X

Fig. 1.2 Classification Tree for the Example Onlineshop Process used in [19]

For each leaf of the classification tree a corresponding snippet of a BPEL-Unit [16, 11] test case is created. Then, given a test input selection, a test case can be generated automatically by composing the corresponding snippets. Constraints on input and output values can be added to the classification tree, so that also assertions can be generated automatically. The classification tree is not only used to configure parameters of initial input messages, but can also be used to specify the contents of intermediate messages.

In current practice, the test selection as in Fig. 1.2 is done by an expert who requires detailed knowledge of the process in order to select a test set that is *effective* and *efficient*, i.e., will successfully detect bugs, and does so with few tests. For an average-size XBP, with thousands of possible tests to choose from, it can take hours to create a good test set; for big processes, with hundreds of thousands of possible tests, the expert easily looses oversight.

When testing with BPELUnit, the tester has some support in assessing the quality of a test suite: it is possible to measure how many of the activity nodes and branches in the process were covered by a test suite. However, such a measure is only available after running the test suite, and so iteratively refining the test suite based on these results is time consuming.

Ultimately, we aim to automate the test selection process. For this purpose, we integrated automatic CTD algorithms in the test case selection framework. In particular, we chose IPOG-C [23] and AETG-SAT [5] for an experimental comparison: IPOG-C is a variant of the IPOG (*In Parameter Order General*) algorithm [10] that is extended to also consider constraints on possible parameter combinations. Similarly, AETG-SAT is an extension of the AETG (*Automatic Efficient Test case Generator*) algorithm [4], that considers constraints on possible parameter combinations, by employing a SAT solver. In our case, the constraints on the possible combinations of inputs are given through the classification tree or additional, manually added constraints, e.g. shipment cannot be successful for unavailable product.

IPOG and AETG take a different approach on selecting tests that yield a full t-wise coverage of the input parameter space. IPOG starts by building all t-tuples of the first t parameters, and then incrementally extends this set horizontally, by including more and more parameters, and vertically, adding more and more tuples as needed, until all t-tuples of all parameters are covered. AETG, on the other hand, uses a heuristic approach for incrementally extending a set of test inputs. In each step, a number of new candidate test inputs is generated, partly randomized, in order to cover as many yet uncovered t-tuples as possible. Then, one of these candidates that covers most uncovered t-tuples is chosen to be included in the test set. This process is repeated until all t-tuples of all parameters are covered. Due to the random component in AETG, the algorithm may produce different results between different runs and the same inputs. IPOG, by contrast, is a deterministic algorithm.

Both algorithms, IPOG and AETG, do not guarantee that the test set is of minimal size. Also, there are no conclusive studies on which approach yields the smallest test set, and therefore, we decided to investigate whether IPOG-C or AETG-SAT creates more efficient test suites in our approach.

1.3 Related Work

The empirical study of of testing techniques is an important subject [3]. The testing of business processes, in particular, is becoming increasingly important, as more and more business and government processes are automated. However, applying CTD for XBPs has not been studied previously; we therefore overview existing work on CTD in related areas.

Kruse et al. [7] and Puoskari et al. [17] studied the effectiveness and efficiency of applying CTD to test an IT management system at IBM. In particular, they also combined a classification-tree-based test selection method with CTD algorithms in a commercial tool, which is now called TESTONA (Assystem Germany GmbH). They compared the classification tree + CTD test design approach with established testing techniques at IBM for the system under study, and concluded that the former can improve the effectiveness of the tests. For measuring the effectiveness of the generated test suites, Kruse et al. measured the test suites' abilities to find manually injected faults. In this paper we instead use activity and branch coverage as a metric for the test suites' effectiveness; as future work, we also plan to use measure the detection rate of faults created via systematic and automatic mutation.

Qi et al. [18] use CTD in combination with automated dynamic exploration testing of web applications. The CTD approach can be used successfully to systematically generate interacting inputs for forms, which accelerate the dynamic exploration.

Kuhn and Reilly [9] conducted an experiment with a browser and server modules, where they ran tests with different t-levels of AETG against the software. Because they knew the number of existing bugs, they could check how many bugs were found with which t-level. For both test sets all bugs were found with $t = 6$. For efficiency reasons, the authors recommended $3 <= t <= 6$ as an advice.

Kuhn et al. [8] did an empirical study for analyzing fault interactions. They queried bug databases of how many conditions influenced a defect. The authors analyzed 7 systems and found that the upper bound of parameter interactions was also 6.

1.4 Experiment Design

1.4.1 Research Questions

Following the goal-question-metric (GQM) method [1], we formulate our research goal as:

The purpose of this study is to *evaluate* the *effectiveness and efficiency of CTD-based testing* from the point of view of *software testers* in the context of *executable business processes*.

When aiming to apply CTD in XBP development projects, the main challenge is to find a suitable t-value and to evaluate the quality of automatically chosen tests by CTD algorithms compared to expert test designs and random tests. The latter serve as a control group in our experiment: random tests are an automated test selection method, which is easy to implement. With some confidence, it can be assumed that random tests will exercise many different cases. As such, we refine our goal into the following two research questions:

1. **What value of t is typically required for executable business processes in order to be effective?** When projects want to use CTD they need to know which configuration of the CTD-algorithms will likely yield the intended results. The higher the t-value is, the more tests are generated and, thus, the more effective the test set will be. But the tests will also run longer, which is a critical factor when tests will be included in continuous builds. However, if the chosen t-value is too low, the tests will likely not be able to detect defects. Therefore, testers need to be able to choose an optimum t-value. Therefore, we want to demonstrate in our experiment, with which t-value most if not all business processes are tested with 100% test coverage measured as (a) basic activity coverage (BPEL equivalent of statement coverage) and (b) decision coverage. These coverage criteria are not a direct measure for the effectiveness of a test suite, but a generally accepted effectiveness metric.
2. **How does the effectiveness and efficiency of the CTD-generated tests, with different t-values, compare to (a) tests created by experts, and (b) a set of tests of equivalent size selected randomly?** When deciding whether to use a CTD-based testing approach, the main question is how such an approach performs compared to a test selection by an expert and to an easy-to-implement random approach. Thus, we compare the efficiency of the test suites generated with IPOG-C and AETG-SAT to those of an expert selection and randomly chosen test suites with the same size.

1.4.2 Case Selection

For studying the effectiveness of the CTD test case selection algorithms, we require a set of XBPs with classification trees. We had access to processes of the Terravis project [2]. Many XBPs in this project are mature, but extensions, improvements and new processes are released frequently [12]. The project applies the (yet new) CTG framework for 5 processes, which we in-

cluded in our experiments. In addition, we used two BPEL processes that were used as examples for the classification tree generator by Schnelle [19] for comparison.

Table 1.1 Classification according to Lübke et al. [14] of the processes in this study (1/2)

	Online Shop	**Credit Approval**	**Land Register Notifications**
Version	-	-	-
Domain	E-Commerce	Banking	Mortgage Transactions
Geography	None	None	Switzerland
Time	2016	2016	2017
Boundaries	-	-	Cross-Organizational
Relationship	No call	No call	Is being called
Scope	Core	Core	Auxilliary
Purpose	Execution	Execution	Execution
People Involvement	None	None	None
Process Language	BPEL 2.0	BPEL 2.0	BPEL 2.0 plus vendor extensions
Execution Engine	Apache ODE[1]		Informatica ActiveVOS 9.2
Model Maturity	Illustrative	Illustrative	Productive
Basic Activities	19	25	84
Structured Activities	8	12	85
Non-linear Struct.A.	6	14	38
Parameters	3^3	$4^1 \cdot 2^4$	$2^7 \cdot 3^7 \cdot 6^1$
Constraints	0	3^1	$3^5 \cdot 2^1 2$
Allowed Configurations	27	52	69888

We compare the different characteristics of the business processes in the classification table (see Tables 1.1 and 1.2) in a templated structured as suggested by Lübke et al. [14].

In addition to the main part of the template, we show static BPEL metrics and the metrics of the classification trees. Static BPEL metrics include the number of basic activities, structured activities, and structured activities without the sequence activity (non-linear structured activities) in order to show the process sizes. The metrics for the classifications are shown following the notation in [10]: the parameters of the classification are notated in the form of x^y, which means that there are y parameters with x values.

1.4.3 Data Collection Procedure

Before analyzing and evaluating the results, we describe the testing process and the tools used for data collection.

Table 1.2 Classification according to Lübke et al. [14] of the processes in this study (2/2)

	Creditor Transfer	**Transfer Approval**	**Approver Process**	**Depot Check**
Version	-	-	-	-
Domain	Mortgage Transactions			
Geography	Switzerland			
Time	2017			
Boundaries	Cross-Organizational			Within-Dep.
Relationship	Calls another			No Call
Scope	Core	Core	Core	Auxilliary
Purpose	Execution	Execution	Execution	Execution
People Involvement	None	Partly	None	None
Process Language	BPEL 2.0 with vendor extensions			
Execution Engine	Informatica ActiveVOS 9.2			
Model Maturity	Productive	Productive	Productive	Productive
Basic Activities	235	33	30	40
Structured Activities	234	34	37	52
Non-linear Struct.A.	71	9	10	16
Parameters	$12^1 \cdot 9^1 \cdot 5^2 \cdot 3^1 \cdot 2^2$	$3^2 \cdot 2^4$	$4^1 \cdot 3^2$	$6^2 \cdot 5^1 \cdot 2^4$
Constraints	$8^1 \cdot 7^2 \cdot 5^4 \cdot 2^{12}$	2^8	2^3	$20^1 \cdot 13^1 \cdot 7^1 \cdot 6^1 \cdot 5^1 \cdot 2^3 \cdot 1^1$
Allowed Configs	7031	27	10	128

Given an XBP to be tested, a test suite is generated as follows. First, the test designer creates the classification tree for the CTG framework as described above, including constraints and underlying BPELUnit fragments.

The classification tree is then be filled out. This can be done manually, a specified number of tests can be automatically generated at random, or a selection can be generated automatically using two alternative CTD algorithms, IPOG-C or AETG-SAT.

The IPOG-C algorithm can be used with the application developed by Vatlin [22], which integrates the IPOG-C implementation of the NIST-ACTS [23] (Advanced Combinatorial Testing System) with the CTG framwork. The AETG-SAT algorithm can be used with the help of the generator developed by Schnelle & Lübke [19].

With a given test selection, the BPELUnit test cases can then be generated and executed.

During generation we measured the size of the test suite and after test case execution we measured the coverage metrics. A BPEL process consists of basic activities as well as structured activities. Basic activities describe the elementary steps of the process and represent single actions. Structured activities determine the control-flow and describe the sequence of activities in BPEL business processes. Structured activities describe conditional and iterative executions of activities.

For the assessment of the used procedure, tools and their interaction, this paper will regard the activity coverage, branch coverage, number of test cases and number of test activities.

Test coverage was measured by analyzing the process logs extracted from the BPEL engine as described by Lübke [13]. We calculated two coverage metrics as described in [15]: (Basic) Activity Coverage calculates the fraction of executed basic activities within a BPEL process. (Conditional) Branch Coverage calculates the fraction of branches taken in a test suite. There are further test coverage metrics defined for BPEL (Handler Coverage and Link Coverage), which we decided not to use because Activity Coverage and Branch Coverage should cover all relevant process elements.

1.4.4 Analysis Procedure

For our study we capture the following metrics for every process project:

- Number of Test Cases, Number of Test Activities, Basic Activity Coverage, and Branch Coverage for the *expert* selected test suite,
- Number of Test Cases, Number of Test Activities, Basic Activity Coverage, and Branch Coverage for the *IPOG-C,* $t \in \{1, 2, 3, 4\}$ generated test suites,
- Number of Test Cases, Number of Test Activities, Basic Activity Coverage, and Branch Coverage for the *AETG-SAT,* $t \in \{2, 3, 4\}$ generated test suites,
- Number of Test Cases, Number of Test Activities, Basic Activity Coverage, and Branch Coverage for the randomly chosen test suites, which have the same size as their IPOG-C, $t \in \{1, 2, 3, 4\}$ counterparts.

For all test strategies with randomness (all randomly chosen test suites and all AETG-SAT test suites), 20 test suites were generated and the mean from the measures was computed in order to account for variances.

For classification trees with fewer than 4 parameters, we could not generate $t = 4$-level test suites (and their random counterparts). This applies to the Online Shop and Approver Process.

From these measurements we compute the efficiency of the tests as $Eff_y(x) = Cov(x)/|y|$ with $x \in \{Activity, Branch\}$ being the coverage metric and $y \in \{TC, TA\}$ being the number of test cases or test activities.

In order to answer our research questions we compared the efficiency of the different strategies. However, we excluded strategies that did not reach at least 75% of the maximum achievable coverage, because we deemed such a low coverage to be too low for practical reasons. Originally we wanted to use a strict 75% threshold. However, we found that depending on the classification tree, not 100% coverage could be reached for all processes. As such we define *maximum coverage* as the maximal coverage value that can be

achieved with the provided classification tree. Test case selection strategies that reach maximum coverage are the most *effective* ones.

1.5 Results

In this section we present the results of our measurement results and their interpretation with regard to our research questions. Last, we assess the validity of our results.

1.5.1 Measurements

The measurements gathered as described above are shown in the boxplots in Fig. 1.3 and Fig. 1.4. Fig. 1.3 shows the activity coverage of the different test suites and Fig. 1.4 shows the branch coverage.

The boxplots aetg*x* and ipog*x* show the values for both AETG-SAT$_{t=x}$ and IPOG-C$_{t=x}$ algorithms. The random-ipog*x* plots show the values of the random test suites with the same number of test cases as the test suite generated with IPOG-C$_{t=x}$.

We can see that the general coverage for both activities and branches is usually above 75% for all strategies. Only IPOG-C$_{t=1}$ and the random test suites often score near or below this threshold. AETG-SAT$_{t=3}$ and IPOG-C$_{t=3}$ however usually score the maximum coverage.

Tables 1.3-1.9 show more detailed information for each XBP.

The data shows that the classifications for the processes "Approver Process" and "Depot Check" are not complete because at least one coverage metric maximizes at below 100%. Interestingly, the "Depot Check" process has full basic activity coverage but misses 100% branch coverage. Upon investigation, it became clear that fault handlers managing exception flow were not triggered which contain basic activities. Because handlers are not included in this test coverage metric, the test gap became only apparent in the basic activity coverage metric.

Table 1.3 Measurements for the **Online Shop** process: Because this process has only 3 parameters, no $t = 4$-level test suites could be generated. 100% Activity and Branch Coverage is reached by all strategies except the random test suites with the same size as IPOG-C$_{t=1}$. Both the expert and IPOG-C$_{t=2}$ selected the most efficient test suite with 9 test cases and 45 test activities each. AETG-SAT created slightly larger test suites for $t = 2$. IPOG-C and AETG-SAT create the same-sized test suites for $t = 3$, which, however, has no improvement over $t = 2$. The expert in this case is not the most efficient one because he designed more test cases than both algorithms with $t = 1$.

Strategy	t-Level	Test Cases	Test Activities	Cov(Act)	Cov(Branch)	Eff-TC(Act)	Eff-TC(Cond)	Eff-TA(Act)	Eff-TA(Cond)
IPOG-C	1	3	15	100.0	100.0	33.33	33.33	33.33	33.33
IPOG-C	2	9	45	100.0	100.0	11.11	11.11	11.11	11.11
IPOG-C	3	27	135	100.0	100.0	3.70	3.70	3.70	3.70
IPOG-C	4	-	-	-	-	0.25	-	-	-
Random I1		3	15	84.2	75.0	28.07	25.00	28.07	25.00
Random I2		9	47	100.0	100.0	11.11	11.11	11.11	11.11
Random I3		27	135	100.0	100.0	3.70	3.70	3.70	3.70
Random I4		-	-	-	-	-	-	-	-
AETG-SAT	2	10	49	100.0	100.0	10.00	10.00	10.00	10.00
AETG-SAT	3	27	135	100.0	100.0	3.70	3.70	3.70	3.70
AETG-SAT	4	-	-	-	-	-	-	-	-
Expert		9	45	100.0	100.0	11.11	11.11	11.11	11.11

Table 1.4 Measurements for the **Credit Approval** process: The expert does not achieve 100% coverage for this process, although IPOG-C with $t \in \{3, 4\}$ and AETG-SAT with $t \in \{2, 3, 4\}$ achieve this goal. Also the random equivalences for IPOG-C $t \in \{3, 4\}$ achieve the same coverage. Interestingly, the most efficient test suite is created by AETG-SAT with $t = 2$. For $t = 3$ AETG-SAT also outperforms IPOG with a smaller test suite on average and even the randomly generated test suites are more efficient than the IPOG-C ones because they use fewer test activities with the same amount of test cases. In this case the expert does not achieve maximum coverage nor the best efficiency value (superseded by the random equivalent for IPOG-C $t = 1$ with regard to branch coverage.)

Strategy	t-Level	Test Cases	Test Activities	Cov(Act)	Cov(Branch)	Eff-TC(Act)	Eff-TC(Cond)	Eff-TA(Act)	Eff-TA(Cond)
IPOG-C	1	4	18	88.0	72.7	22.00	18.18	22.00	18.18
IPOG-C	2	8	36	96.0	90.9	12.00	11.36	12.00	11.36
IPOG-C	3	24	111	100.0	100.0	4.17	4.17	4.17	4.17
IPOG-C	4	40	180	100.0	100.0	2.50	2.50	2.50	2.50
Random I1		4	18	80.0	77.3	20.00	19.32	20.00	19.32
Random I2		8	36	94.0	90.9	11.75	11.36	11.75	11.36
Random I3		24	106	100.0	100.0	4.17	4.17	4.17	4.17
Random I4		40	170	100.0	100.0	2.50	2.50	2.50	2.50
AETG-SAT	2	10	42	100.0	100.0	10.00	10.00	10.00	10.00
AETG-SAT	3	22	99	100.0	100.0	4.55	4.55	4.55	4.55
AETG-SAT	4	41	182	100.0	100.0	2.44	2.44	2.44	2.44
Expert		4	18	88.0	72.7	22.00	18.18	22.00	18.18

Table 1.5 Measurements for the **Land Register Notifications** process: The expert scores best with 100% coverage for both measures and the fewest test cases (25) for this coverage level. IPOG-C and AETG-SAT with $t \in \{2, 3, 4\}$ also achieve 100% coverage for both measures, which is not reached by any random test suite. Those fail with regard to branch coverage. IPOG-C and AETG-SAT require more test cases than the expert. IPOG-C creates a smaller test suite than AETG-SAT for $t \in \{2, 4\}$, while for $t = 3$ AETG-SAT is more efficient.

Strategy	t-Level	Test Cases	Test Activities	Cov(Act)	Cov(Branch)	Eff-TC(Act)	Eff-TC(Cond)	Eff-TA(Act)	Eff-TA(Cond)
IPOG-C	1	6	29	72.6	66.7	12.10	11.11	12.10	11.11
IPOG-C	2	30	140	100.0	100.0	3.33	3.33	3.33	3.33
IPOG-C	3	125	593	100.0	100.0	0.80	0.80	0.80	0.80
IPOG-C	4	401	1959	100.0	100.0	0.25	0.25	0.25	0.25
Random I1		6	33	38.1	35.4	6.35	5.89	6.35	5.89
Random I2		30	162	83.3	76.8	2.78	2.56	2.78	2.56
Random I3		125	684	100.0	93.0	0.80	0.74	0.80	0.74
Random I4		401	2217	100.0	97.7	0.25	0.24	0.25	0.24
AETG-SAT	2	32	152	100.0	100.0	3.08	3.08	3.08	3.08
AETG-SAT	3	124	600	100.0	100.0	0.81	0.81	0.81	0.81
AETG-SAT	4	410	2006	100.0	100.0	0.24	0.24	0.24	0.24
Expert		25	112	100.0	100.0	4.00	4.00	4.00	4.00

Table 1.6 Measurements for the **Creditor Transfer** process: Both IPOG-C and AETG-SAT achieve 100% activity and branch coverage for $t \in \{3, 4\}$ but both also fail to achieve this coverage with $t = 2$. Interestingly, the expert achieves 100% coverage with less test cases than both $t = 2$-level algorithms and thus is the most efficient one to reach maximum coverage.

Strategy	t-Level	Test Cases	Test Activities	Cov(Act)	Cov(Branch)	Eff-TC(Act)	Eff-TC(Cond)	Eff-TA(Act)	Eff-TA(Cond)
IPOG-C	1	14	133	59.6	59.3	4.26	4.23	4.26	4.23
IPOG-C	2	68	768	99.6	90.7	1.46	1.33	1.46	1.33
IPOG-C	3	268	3151	100.0	100.0	0.37	0.37	0.37	0.37
IPOG-C	4	934	11024	100.0	100.0	4.35	4.35	4.35	4.35
Random I1		14	164	49.6	50.9	3.54	3.64	3.54	3.64
Random I2		68	834	73.5	74.1	1.08	1.09	1.08	1.09
Random I3		268	3244	80.3	78.7	0.30	0.29	0.30	0.29
Random I4		933	11083	93.0	91.7	0.10	0.10	0.10	0.10
AETG-SAT	2	67	747	99.6	90.7	1.49	1.35	1.49	1.35
AETG-SAT	3	279	3234	100.0	100.0	0.36	0.36	0.36	0.36
AETG-SAT	4	968	11434	100.0	100.0	0.10	0.10	0.10	0.10
Expert		35	741	100.0	100.0	2.86	2.86	2.86	2.86

Table 1.7 Measurements for the **Transfer Approval** process: The expert missed the 100% coverage goal with this process and scored like IPOG-C$_t$ = 1 97% Activity Coverage and 85.7% Branch Coverage. All random test suites miss the 100% target as well. IPOG-C and AETG-SAT with levels $t \in \{2, 3, 4\}$ all reach 100% Activity Coverage and Branch Coverage. Both IPOG-C and AETG-SAT with $t = 2$ create the most efficent test suite with 8 test cases and 61 test activities. Coverage cannot increase with higher t-levels. However, AETG-SAT creates smaller test suites than IPOG-C with these levels.

Strategy	t-Level	Test Cases	Test Activities	Cov(Act)	Cov(Branch)	Eff-TC(Act)	Eff-TC(Cond)	Eff-TA(Act)	Eff-TA(Cond)
IPOG-C	1	4	26	97.0	85.7	24.24	21.43	24.24	21.43
IPOG-C	2	8	61	100.0	100.0	12.50	12.50	12.50	12.50
IPOG-C	3	15	122	100.0	100.0	6.67	6.67	6.67	6.67
IPOG-C	4	23	194	100.0	100.0	4.35	4.35	4.35	4.35
Random I1		4	31	81.8	71.4	20.45	17.86	20.45	17.86
Random I2		8	69	84.8	78.6	10.61	9.82	10.61	9.82
Random I3		15	126	90.9	100.0	6.06	6.67	6.06	6.67
Random I4		23	195	100.0	100.0	4.35	4.35	4.35	4.35
AETG-SAT	2	8	61	100.0	100.0	12.50	12.50	12.50	12.50
AETG-SAT	3	12	95	100.0	100.0	8.33	8.33	8.33	8.33
AETG-SAT	4	22	190	100.0	100.0	4.44	4.44	4.44	4.44
Expert		4	28	97.0	85.7	24.24	21.43	24.24	21.43

Table 1.8 Measurements for the **Approver Process**: Because this process has only 3 parameters, no 4-level test suites could be generated. Except for IPOG-C$_t$ = 1 and its random counterpart, which reach 93.33% activity coverage and around 50% branch coverage, all strategies deliver 100% test coverage and 75% branch coverage. With 6 Test Cases and 32 Test Activities the expert has selected the most efficient test suite that delivers the same coverage like the remaining strategies but with fewer test cases. IPOG-C and AETG-SAT create test suites with the same amount of test cases and test activities for this process project.

Strategy	t-Level	Test Cases	Test Activities	Cov(Act)	Cov(Branch)	Eff-TC(Act)	Eff-TC(Cond)	Eff-TA(Act)	Eff-TA(Cond)
IPOG-C	1	4	14	93.3	50.0	23.33	12.50	23.33	12.50
IPOG-C	2	10	42	100.0	75.0	10.00	7.50	10.00	7.50
IPOG-C	3	10	42	100.0	75.0	10.00	7.50	10.00	7.50
IPOG-C	4	-	-	-	-	4.35	-	-	-
Random I1		4	15	93.3	50.0	23.33	12.50	23.33	12.50
Random I2		10	42	100.0	75.0	10.00	7.50	10.00	7.50
Random I3		10	42	100.0	75.0	10.00	7.50	10.00	7.50
Random I4		-	-	-	-	-	-	-	-
AETG-SAT	2	10	42	100.0	75.0	10.00	7.50	10.00	7.50
AETG-SAT	3	10	42	100.0	75.0	10.00	7.50	10.00	7.50
AETG-SAT	4	-	-	-	-	-	-	-	-
Expert		6	32	100.0	75.0	16.67	12.50	16.67	12.50

Table 1.9 Measurements for the **Depot Check** process: The maximum coverage for activities are 85% meaning that the classification tree is not complete. Both algorithms regardless of the t-value reach the maximum activity coverage. This is also true for the random test suites equivalent in size to the IPOG-C $t > 1$ suites. Only the random test suite corresponding to IPOG-C $t = 1$ does not achieve maximum activity coverage. IPOG-C $t = 2$ and the expert additionally miss the maximum branch coverage. However, the expert created the most efficient test suite with regard to branch coverage, while the random test suite relating to IPOG-C $t = 1$ is the most efficient test suite with regard to activity coverage.

Strategy	t-Level	Test Cases	Test Activities	Cov(Act)	Cov(Branch)	Eff-TC(Act)	Eff-TC(Cond)	Eff-TA(Act)	Eff-TA(Cond)
IPOG-C	1	7	45	85.0	79.0	12.14	11.28	12.14	11.28
IPOG-C	2	35	194	85.0	94.7	2.43	2.71	2.43	2.71
IPOG-C	3	81	450	85.0	100.0	1.05	1.23	1.05	1.23
IPOG-C	4	128	726	85.0	100.0	0.66	0.78	0.66	0.78
Random I1		7	38	81.2	77.8	11.61	11.11	11.61	11.11
Random I2		35	193	85.0	100.0	2.43	2.86	2.43	2.86
Random I3		81	454	85.0	100.0	1.05	1.23	1.05	1.23
Random I4		128	726	85.0	100.0	0.66	0.78	0.66	0.78
AETG-SAT	2	35	195	85.0	100.0	2.43	2.86	2.43	2.86
AETG-SAT	3	77	427	85.0	100.0	1.10	1.30	1.10	1.30
AETG-SAT	4	128	726	85.0	100.0	0.66	0.78	0.66	0.78
Expert		8	43	85.0	89.5	10.62	11.18	10.62	11.18

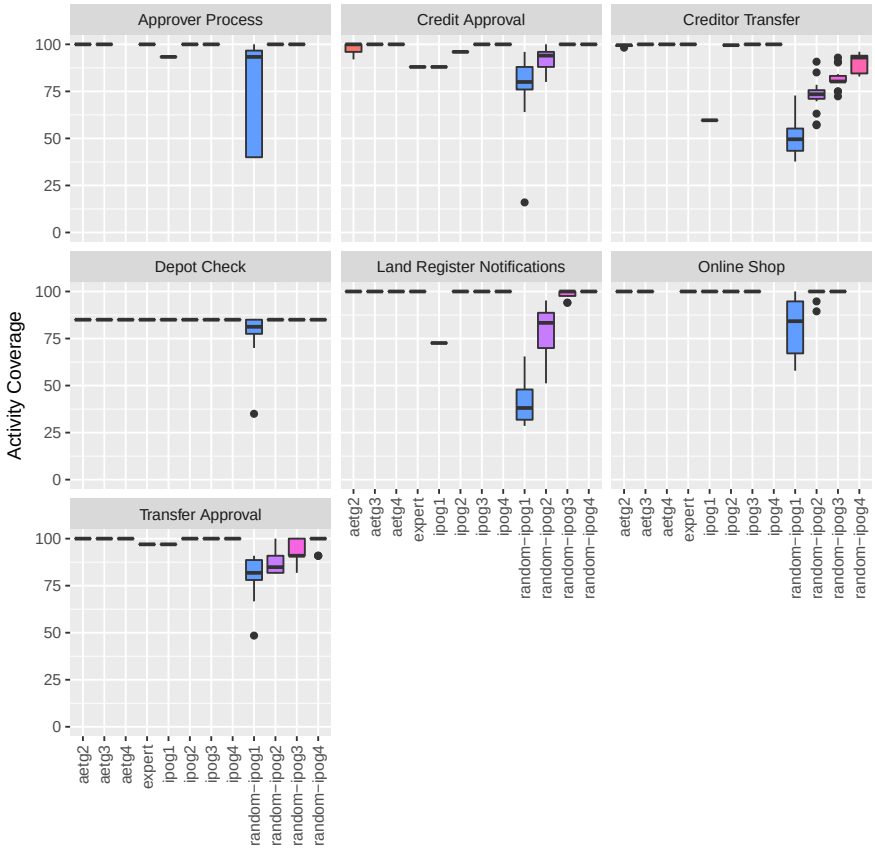

Fig. 1.3 Activity Coverage of different Algorithms for the analyzed Processes

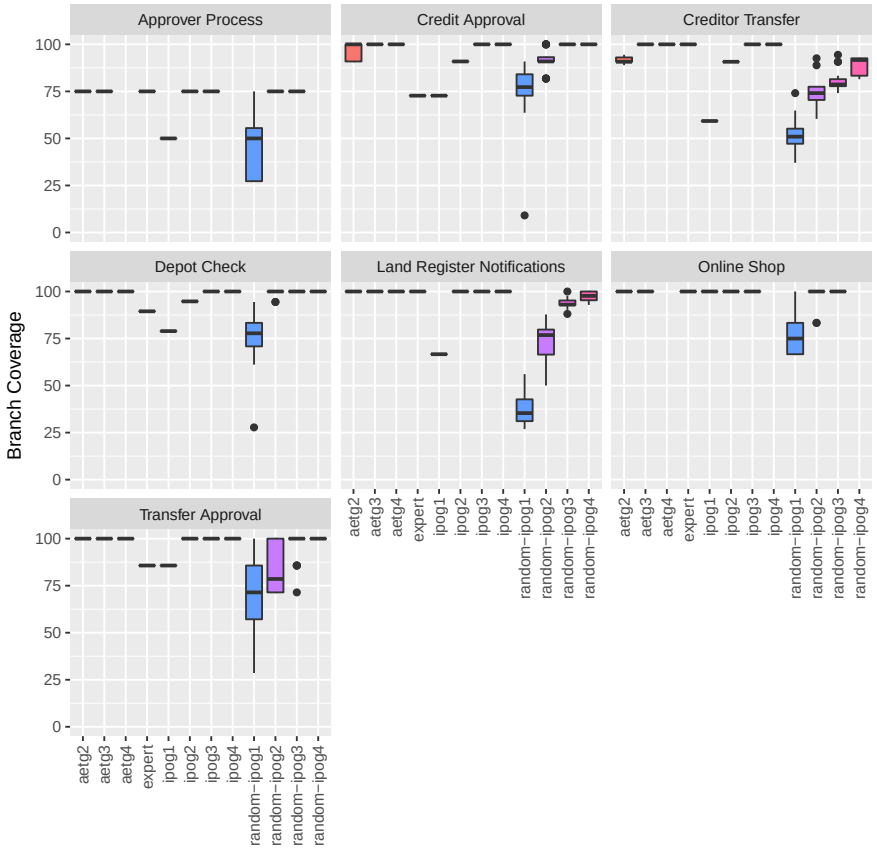

Fig. 1.4 Conditional Branch Coverage of different Algorithms for the analyzed Processes

1.5.2 Interpretation

RQ1 (Required t-Value): For all processes and the given classification trees that are part of our experiment, both IPOG-C as well as AETG-SAT generate the maximum coverage at $t = 3$. Higher t-values have offered no additional coverage benefits in our dataset. From an efficiency point of view, $t = 2$ yields more effective results with often maximum coverage and in other cases only a small penalty while requiring only a fraction of test cases compared to $t = 3$. No process profited from a further increase of t to $t = 4$; the test suites become very large without any coverage benefit.

This is in contrast to other types of software (see [9, 8]), where some defects in certain kinds of applications are only triggered with $t = 6$. This is a significant difference because less test cases are required for XBPs to fully cover the software than with "traditional" software.

RQ2 (Efficiency and Effectiveness): Looking at the data, IPOG-C and AETG-SAT deliver comparable test coverage for the same t-level. Sometimes one algorithm scores better, sometimes the other. Better performance is not bound to any particular process. For one t-level for one given process, one algorithm can score better than the other, while scoring worse for another t-level on the same process.

Random test case selection scores worse with regard to both effectiveness and efficiency across our XBP set, so structured test selection with IPOG-C and AETG-SAT beats pure randomness. Also IPOG-C$_{t=1}$ is not very effective, but on that low level of effectiveness it is quite efficient.

For industry XPBs, the expert is the most efficient "strategy" – especially because the expert selection usually reaches maximum possible coverage. Sometimes, however, even the expert fails to achieve the maximum coverage although this is possible with the given classification tree. Thus, while being more efficient than IPOG-C and AETG-SAT for the same test suite size, the expert is not as effective as the automatic test case selection algorithms.

Both algorithms and experts cannot perform better than the input data: the testers for three industry processes failed to provide a classification tree that is sufficient to reach 100% coverage.

During our analysis we observed one property of the current test coverage metric definitions for BPEL that is distinctively different from the properties for other programming languages: The power of branch coverage compared to the power of activity coverage. Usually branch coverage is stricter than activity coverage (called statement coverage for programming languages.) However, BPEL allows the modeling of several handler types that are modeled outside the main control-flow and are thus not included in the branch coverage definition – test managers should be aware of this difference. We observed a lower activity coverage measure in the Depot Check process, where it was possible to reach 100% branch coverage (e.g., with AETG-SAT$_{t=2,3}$ and IPOG-C$_{t=2,3}$) but the maximum activity coverage was 85%. Our anal-

ysis revealed that this process contains further basic activities in two fault handlers. Because all branches in the normal process flow were triggered, branch coverage was 100%, but the basic activities in the handlers were not executed, thereby lowering activity coverage below 100%.

1.5.3 Evaluation of Validity

Our data set only contains processes from one project and is quite small: we covered 5 industrial XBPs and 2 synthetic ones. Also for all XBPs only one expert selected test cases manually, and these were the first processes for which the CTG framework was applied. Therefore, the results that we obtain for the expert choices may not be representative.

However, our data set consisted of XBPs of very different sizes, with different numbers of constraints and classification tree sizes. Therefore, we think that our results are still generalizable with regards to chosen t-values and efficiency.

1.6 Conclusion and Future Work

Our work demonstrates that CTD can be successfully applied to the domain of executable business processes. We developed a novel testing technique for executable business processes that combines automatic CTD algorithms with automatic BPELUnit test generation based on classification trees. We conducted experiments with industrial processes that indicate that CTD algorithms can replace the expert in the selection of the test cases. For the processes that we considered, a t-level of 3 for both IPOG-C and AETG-SAT was always sufficient to reach the maximum coverage that was possible with a given classification tree.

IPOG-C and AETG-SAT deliver comparable efficiency for a given t-level. For some processes IPOG-C created smaller test suites than AETGT-SAT and vice versa.

However, the expert could create test suites that were more efficient, i.e., they required less test cases for a given efficiency. The expert also missed the maximum coverage in some processes. All in all, the automatic selection of test cases is more reliable but a bit less efficient.

One interesting finding is that in our experiments coverage for XBPs maximizes at $t = 3$ while in other studies the coverage for some types of software systems maximizes with t up to 6. If further studies strengthen this finding, an evaluation of differences in the structure of the software should be done to explain this significant difference.

From the practitioners' point of view, our conclusion is follows. The implementation of the described automated CTD technique within the Terravis project was successful and our experiments showed that automatic selection of test cases from test classification trees with IPOG-C and AETG-SAT meet the requirements of practice.

Especially the huge number of constraints requires algorithms that consider these constraints and can calculate test cases quickly. Both IPOG-C and AETG-SAT can handle the classification trees of the studied project well. For all studied processes, $t = 3$-level covered the possible maximum of activities and branches in the processes. $t = 2$-level generates less test cases and covers more activities/branches per test case but failed to reliably cover all activities and branches.

Regarding industry usage, test managers need to decide whether efficiency or maximizing coverage are more important.

While IPOC-C and AETG-SAT are comparable for generating test case selections, due to the random component within AETG, the result is non-deterministic and coverage varies between different generation runs, which might be problematic in industry projects that cannot afford to re-run tests multiple times but want to save time.

While the expert was able to create more efficient test case selections than both IPOG-C and AETG-SAT, the expert sometimes missed test cases to create full coverage. The task of selecting test cases is especially burdensome if the classification tree becomes large, which is the case with the executable processes in our study. Therefore, we think that under most circumstances, the added efficiency by the tester does not outweigh the required effort. However, the exact characterization of this topic and the required considerations are left to future work.

An important aspect is the completeness of the classification tree that must be created manually by the testers: We observed that for some executable processes the classification tree was incomplete and thus neither the expert nor any algorithm could achieve 100% coverage. Industry projects require methods for detecting such problems. In this regard, an automated CTD-based approach can even help pinpoint the problem: If coverage is below 100% for an automatically generated test set for $t = 3$, this is a strong indicator that the problem is an incomplete classification tree. By contrast, a test expert who is in an hour-long process of improving the test selection to increase coverage may realize only late that the classification tree is the problem. Both approaches benefit from using coverage measurement tools in order to detect missing cases on a technical level and fix this by adding business-driven equivalence classes to the classification tree.

1.6.1 Future Work

While we are happy that we could conduct an experiment with industrial implementations, such possibilities are yet too rare. We would especially like to see (or helping doing) case studies that can be replicated by others. Having an even larger executable business process sets is desirable as well.

Replicated case studies will hopefully strengthen our findings but will also enable better predictions on which t-levels for the different algorithms are appropriate to reach maximum coverage. Future research should identify properties of processes or heuristics that can predict which t-level is likely sufficient for achieving maximum coverage for a given classification tree beforehand.

We have also seen that the expert selects more efficient test cases for executable business processes than the algorithms do. Further research into why this is the case and extensions or optimizations of the existing algorithms are a valuable research target in order to further enhance the efficiency of generated test suites.

Furthermore, we have seen that the current definitions of BPEL test coverage are not totally satisfactory: In order to know whether everything process-flow related has been covered during testing, both basic activity coverage and branch coverage are required. Extending branch coverage by handler coverage would eliminate the problem and make the new metric more powerful than basic activity coverage. However, even if such an extended metric, the data-flow code hidden in the executable business processes is not taken into account. Due to this, we would like to proceed further by incorporating test coverage metrics that also take the data-flow into account into our research. Possible effectiveness metrics could be derived by using systematically seeded faults (mutations) in the process and data-flow dependent metrics.

References

1. Victor R Basili. Applying the goal/question/metric paradigm in the experience factory. *Software Quality Assurance and Measurement: A Worldwide Perspective*, pages 21–44, 1993.
2. Walter Berli, Daniel Lübke, and Werner Möckli. Terravis – large scale business process integration between public and private partners. In Erhard Plödereder, Lars Grunske, Eric Schneider, and Dominik Ull, editors, *Lecture Notes in Informatics (LNI), Proceedings INFORMATIK 2014*, volume P-232, pages 1075–1090. Gesellschaft für Informatik e.V., Gesellschaft für Informatik e.V., 2014.
3. L. C. Briand. A critical analysis of empirical research in software testing. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 1–8, Sept 2007.
4. David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The aetg system: An approach to testing based on combinatorial design. *IEEE Trans. Softw. Eng.*, 23(7):437–444, July 1997.

5. Myra B Cohen, Matthew B Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, 34(5):633–650, 2008.
6. M. Grochtmann. Test case design using classification trees. In *Proceedings of the International Conference on Software Testing Analysis & Review (STAR 1994), Washington D.C., USA*, 1994.
7. Peter Kruse, Onn Shehory, Daniel Citron, Nelly Condori-Fernández, Tanja Vos, and Bilha Mendelson. Assessing the applicability of a combinatorial testing tool within an industrial environment, 01 2014.
8. D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, June 2004.
9. D Richard Kuhn and Michael J Reilly. An investigation of the applicability of design of experiments to software testing. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 91–95. IEEE, 2002.
10. Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun, and James Lawrence. Ipog-ipog-d: Efficient test generation for multi-way combinatorial testing. *Softw. Test. Verif. Reliab.*, 18(3):125–148, September 2008.
11. Daniel Lübke. Test and Analysis of Service-Oriented Systems. chapter Unit Testing BPEL Compositions. Springer, 2007.
12. Daniel Lübke. Using Metric Time Lines for Identifying Architecture Shortcomings in Process Execution Architectures. In *Software Architecture and Metrics (SAM), 2015 IEEE/ACM 2nd International Workshop on*, pages 55–58. IEEE, 2015.
13. Daniel Lübke. Calculating test coverage for bpel processes with process log analysis. In *Proceedings of the Eighth International Conference on Business Intelligence and Technology (BUSTECH 2018)*, 2018 (accepted).
14. Daniel Lübke, Ana Ivanchikj, and Cesare Pautasso. A template for sharing empirical business process metrics. In *Business Process Management Forum - BPM Forum 2017*, 2017.
15. Daniel Lübke, Leif Singer, and Alex Salnikow. Calculating BPEL Test Coverage through Instrumentation. In *Workshop on Automated Software Testing (AST 2009), ICSE 2009*, 2009.
16. Philip Mayer and Daniel Lübke. Towards a BPEL Unit Testing Framework. In *TAV-WEB '06: Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications, Portland, USA*, pages 33–42, New York, NY, USA, 2006. ACM Press.
17. Elisa Puoskari, Tanja E. J. Vos, Nelly Condori-Fernandez, and Peter M. Kruse. Evaluating applicability of combinatorial testing in an industrial environment: A case study. In *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to Testing Automation*, JAMAICA 2013, pages 7–12, New York, NY, USA, 2013. ACM.
18. Xiao-Fang Qi, Zi-Yuan Wang, Jun-Qiang Mao, and Peng Wang. Automated testing of web applications using combinatorial strategies. *Journal of Computer Science and Technology*, 32(1):199–210, Jan 2017.
19. Thilo Schnelle and Daniel Lübke. Towards the generation of test cases for executable business processes from classification trees. In *Proceedings of the 9th Central European Workshop on Services and their Composition (ZEUS) 2017*, 2017.
20. Kuo-Chung Tai and Yu Lei. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, Jan 2002.
21. K. Tatsumi. Test case design support system. In *Proceedings of International Conference on Quality Control, Tokyo*, pages 615–620, 1987.
22. David Vatlin. Generation of Test Suites for Business Processes based on Combinatorial Test Design. Bachelor thesis, Leibniz University of Hanover, Germany, November 2017.

23. L. Yu, Y. Lei, M. Nourozborazjany, R. N. Kacker, and D. R. Kuhn. An efficient algorithm for constraint handling in combinatorial test generation. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 242–251, March 2013.