Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 1585–1590
DOI: 10.15439/2014F263
ACSIS, Vol. 2

Automating Test Case Design within the Classification Tree Editor

Ute Zeppetzauer
Berner and Mattner Systemtechnik GmbH
Munich, Germany
Email: ute.zeppetzauer@berner-mattner.com

Peter M. Kruse
Berner and Mattner Systemtechnik GmbH
Berlin, Germany
Email: peter.kruse@berner-mattner.com

***Abstract*—This paper describes how the proven test design technique of the classification tree method is extended within the classification tree editor in order to contribute to current test design matters. The classification tree editor not only provides the tooling to use the method but also to apply new and helpful features in the test design process. This includes the automatic generation of test cases and test sequences according to desired test depth and focus, automated boundary value analysis, various tool couplings to integrate in each individual test process and supporting features like test evaluation or test coverage analysis amongst others.**

***Keywords*-classification tree method; classification tree editor; combinatorial interaction testing;**

I. Introduction

THE classification tree method [1] as well as the editor [2] have been developed at Daimler's research department for software technology in the 90ties. In the last 20 years both, method and tool became proven in practice around the world.

The classification tree method as a black box test design technique was introduced by Matthias Grochtmann and Klaus Grimm in 1993 [1]. The basic idea of the classification tree method is to separate the input data characteristics of the system under test into different classes that directly reflect the relevant test scenarios. The descriptive method covers the categories of test case design needed in industry, such as *equivalence class tests*, *boundary value analysis* [3], *interface testing*, as well as *combinatorial* [4] and *statistical testing* [5].

In the last 6 years where Berner and Mattner took over the further development of the classification tree method and the editor, it got extended by academic research results and industrial input. This includes additional statistical testing methods [5], a professional requirements tracing [6], a synchronization to test management, test evaluation, coverage analysis or support for website testing [7]. The classification tree editor supports thus the tester in the test design phase. It offers multiple functions to structure the test problem, to systematically generate test scenarios and to do all this within the scope of the testers environment.

The outline of this paper is as follows: Section II introduces the classification tree method, while Section III shows how it will prove to serve testers needs for a more automated way in designing test cases within the classification tree editor. Section IV the integration into the test process with requirements and test management tools are introduced. Section V details Excel import and combinatorial test coverage analysis, test evaluation is given in Section VI. Related work can be found in Section VII, while conclusion is drawn in Section VIII.

II. Test Case Design using the Classification Tree Method

A. How to create a classification tree

The basic idea of the classification tree method is to separate the input data characteristics of the system under test into different classes that directly reflect the relevant test scenarios (*classifications*) [1]. The main source of information is the specification of the system under test or a functional understanding of the system should no specification exist. Two significant steps must be performed to create a classification tree:

- The identification of relevant factors involves the determination and structuring of the relevant test scenarios and their interrelations to other parts of the system under test.
- The test specifications combine the relevant factors needed in order to achieve the desired test coverage.

The first phase begins with the identification of the classifications that are relevant for testing based on a functional description and understanding of the system under test. For each classification, there may be several input data that are all to be considered during testing, the *classes* of a classification.

Each classification should have a limited number of clearly defined input scenarios for the system under test. The input data characteristics are used to define the input ranges required for the relevant test scenarios. This is a classification in the mathematical sense: The set of all possible inputs is disjointly and completely classified into subsets - the classes. The separation based on the input data characteristics is done independently for each test scenario, and can therefore be done easily.

This approach applies the concept of equivalence class testing [3]: Testing with a data item that is representative of an equivalence class makes tests with all other elements of the same class redundant and therefore unnecessary because

978-83-60810-58-3/$25.00

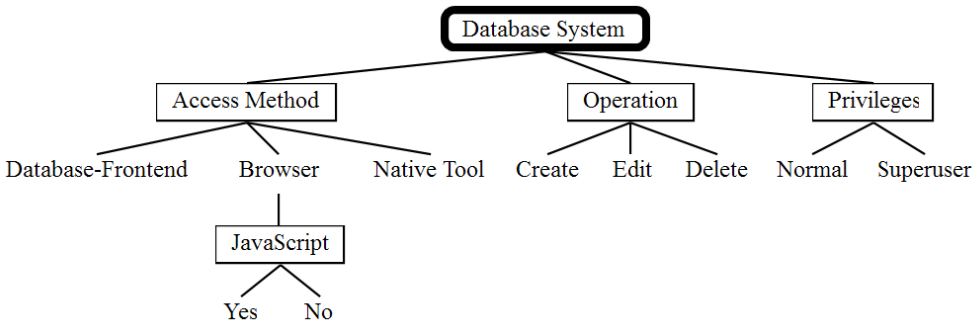

Fig. 1. Test Object *Database Management System*

there should be no difference in how they are handled during execution of the system under test.

Example. Figure 1 shows a classification tree for a database management system. Three aspects of interest (*Access Method*, *Operation* and *Privileges*) have been identified for the system under test. The classifications are partitioned into classes which represent the partitioning of the concrete input values. In our example the refinement aspect *JavaScript* is identified for the class *Browser* and it is divided further into two classes *Yes* and *No*.

B. Dependency Rules

Often, there are dependencies or constraints [8] between some classes of the classification tree. To overcome this design gap, the CTE offers to describe the relationships between the elements of the classification tree by defining dependency rules. The classification tree editor provides two mechanisms for defining dependency rules:

1) Logical dependency rules between the classes of a classification tree using propositional logic. The result is that test cases that would not fulfill the previously defined rules are not generated and vice versa [9].
2) Numerical dependencies between the classifications using logical and numerical operators. They serve to express mathematical dependencies between the elements of the classification tree [10].

C. Boundary Value Analysis

The boundary value analysis [3] is a helpful tool to run an analysis automatically with user specified input. There are two ways of applying the boundary value analysis within the CTE. The first is for an initial analysis, the second is to analyze and expand a classification tree with more possible parameters and intervals see Fig. 2.

For the initial analysis, the CTE supports to create parameters and intervals. For each interval it is possible to set the boundary borders so that CTE will create the corresponding boundary values. The classification tree is created from the specified values.

For the analysis or expansion of the tree, the CTE loads the existing data into its own data model. Then it is possible to add new parameters and intervals. The CTE then generates the boundary values and adds additional elements to the tree.

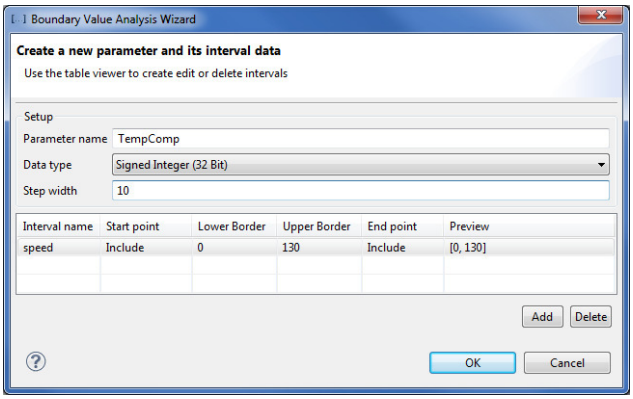

Fig. 2. Parameter and interval settings in CTE

III. Test Case Generation Facilities within the CTE

In practice, not every test generation mechanism is applicable to every test problem. Thus, there are numerous facilities available within the classification tree editor (CTE) in order to serve testers in many ways. The following test case generation mechanisms are implemented in the CTE.

- combinatorial test case generation
- prioritized test case generation
- test sequence generation

A. Combinatorial Test Case Generation

For combinatorial testing, the question is on how to achieve adequate test coverage without having to test too much. Complex software testing problems easily reach a maximum number of test cases of several billions. This is not feasible to handle. So a wise, structured and reproducible selection of test cases according to the current test problem saves effort, time and helps to actually better understand the testing focus. For this, combinatorial testing with all of its aspects is ideal. It brings variation to a test suite with a clear definition of the test intensity.

The CTE offers combinatorial test case generation facilities that support the structured variation [5], [9]. This includes pairwise combination, threewise combination, minimal test coverage and individual combinatorial rules.

The minimal combination $A + B$ ensures that each class of classification A and each class of classification B is considered in at least one test case.

The pairwise combination $(A_1, \ldots, A_n)$ ensures that each class of the classifications A_1 to A_n is combined pairwise with every other class in at least one test case.

The threewise combination $(A_1, \ldots, A_n)$ ensures that every possible combination of three classes of the different classifications A_1 to A_n is generated in at least one test case.

Higher strength interaction coverage can be achieved by practical testing approaches [11]. The key to unlocking better performance for higher strengths of interaction, seems to rely upon the use of constraints, which reduce the number of possibilities to be considered. In CTE, constraints are defined in terms of dependency rules.

Individual combination rules may be designed upon the needs of each test problem. All standard operators are available for that.

Logical expressions are used to formulate dependency rules. The CTE XL allows to specify any kind of logical dependency rules, containing {*AND*, *OR*, *NOT*, $\Rightarrow$, $\Leftrightarrow$, *XOR*, *NOR*, *NAND* }. Parentheses are used to formulate more complex expressions [9].

B. Prioritized Test Case Generation

In addition to the above mentioned classical combinatorial test case generation, the CTE offers the possibility to add weights to classes for a prioritized test case generation [5].

Weights on classes can be used to create test cases in an order corresponding to their relevance. For example, those tests that have revealed most of the failures in previous runs can be executed more frequently.

Within CTE, weights are distributed to classes in order to generate prioritized test cases according to occurrence, error or risk probability. The result is a test suite of tests covering the pairwise combination criterion and being sorted according to their relevance most relevant test cases at the top of the test suite, least relevant at the bottom [12]. By using the optimize menu, the tester can adjust the test suite individually by selecting the weighted coverage see Fig. 3.

C. Test Sequence Generation

Many software-based systems are state-based. Thus, test data used in test steps of one test sequence must provide a logical sequence to run the desired state transitions. The manual modeling of test sequences important for testing is a challenging task for the tester. Within the CTE this can be done automatically [13].

The tester defines simple state machines [14], modeling the behavior of the system see Fig. 4. Test cases are then derived from that.

A possible application for Hardware-in-the-loop testing has been discussed recently [15].

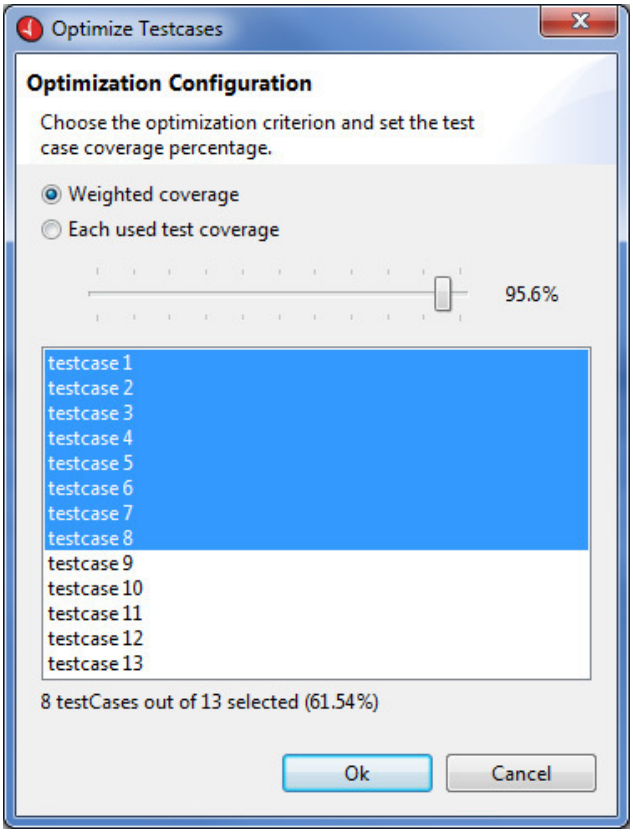

Fig. 3. Optimizing test suites according to weighted coverage in CTE

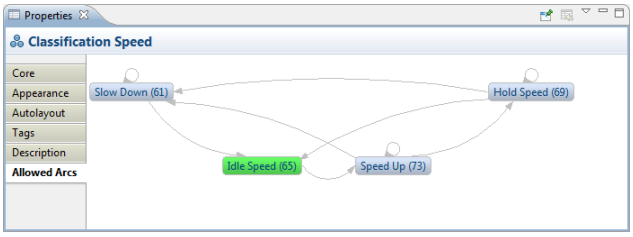

Fig. 4. Allowed arcs for the example classification speed in CTE

IV. Requirements Tracing and Test Management

A. Requirements Tracing within CTE

In the development process, test design is not the first action to take. Before even thinking about testing, the function must be specified. For this, there are several requirements management tools on the market which serve to define and monitor specifications.

For test design, a connection to those requirements might be a huge advantage with regard to traceability. Also, when linking requirements to test cases, gaps can be detected.

The CTE offers the tester to link tree elements and test cases to requirements from MS Access or IBM Rational DOORS [6]. The result is a requirements matrix that directly shows where there is still work to do (Fig. 5). Linked and not yet used requirements can be visualized.

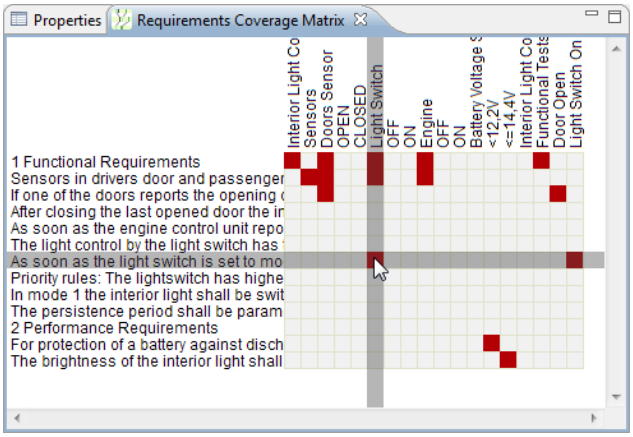

Fig. 5. Requirements matrix view in CTE showing linked CTE objects

Often, requirements change in the process. For this, synchronizing the changed database shows the tester in CTE where to modify the classification tree in order to be conform to the specification [6].

This connection is currently one way to get information from DOORS to use in CTE. In several practical projects, a bidirectional connection has been established, where the results from CTE were exported to the corresponding DOORS module.

B. Test Management in CTE

Parallel to all activities in the test process, test management is essential. For this, CTE synchronizes with HP ALM to commit or submit test cases from and to the test management tool. All information relevant to the tester will be downloaded from the server and also uploaded with updated data. This is supported by a comprehensive mapping mechanism in CTE (Fig. 6). The classification tree will be saved to the central folder on the server in order to make it available for other users for systematic testing.

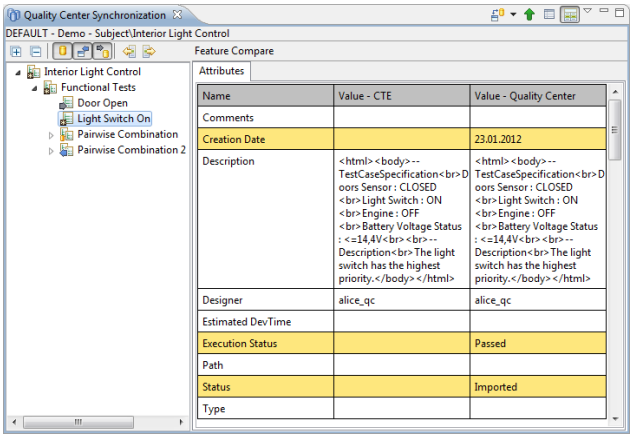

Fig. 6. CTE view of the HP ALM synchronization status after adding new values

Currently, the synchronization is made with the test plan module. For future work, it is planned to extend that connection to the requirements information in order to have a consistent and traceable process, and to extend it to read the information from the test lab which shows information to evaluate the performed tests.

V. Excel Import and Coverage Analysis

A. Excel Import

The most common tool for test design is Excel. There are many different ways in practice to use it within the testing process. With this background, an Excel import to continue with a systematic testing approach in essential.

All Excel sheets no matter if the column or the rows represent the test cases are imported into CTE, as well as the already defined test cases. So the classification tree is built, the test cases are imported and further combinatorial testing can be performed.

B. Coverage Analysis

In practice, the need to give a value for coverage increases. The tester needs to know how *good* the defined test cases are. Within the CTE, the coverage analysis gives a basic overview of the covered tuples see Fig. 7.

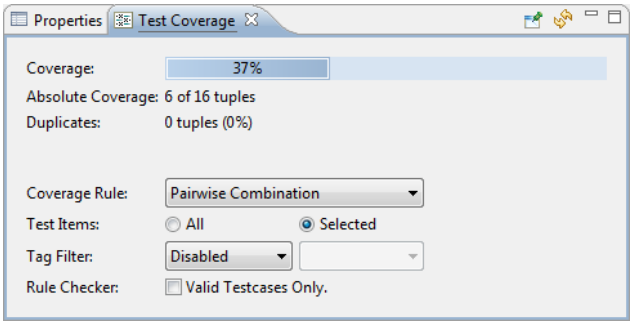

Fig. 7. Test coverage of a set of existing test cases to the coverage rule pairwise combination

If we refer to the Excel import above, it is hard to tell how good or bad the manually defined test cases that have been imported are. With the coverage analysis function, these imported test cases can be compared to specific generation criteria like the pairwise rule, or the minimal coverage rule. According to the result, test suites can then be complemented in order to reach the desired coverage criteria.

VI. Test Evaluation

Throughout the test process, test evaluation is the final step to determine the relevance of the test results. Within CTE, test results for test cases can be either imported or added and then evaluated. The CTE can track test results in terms of *Passed*, *Failed*, *Error*, *Not Executed*. By means of a root cause analysis, the error rate of single classes can be detected and thus gives important information on possible defects of the system under test.

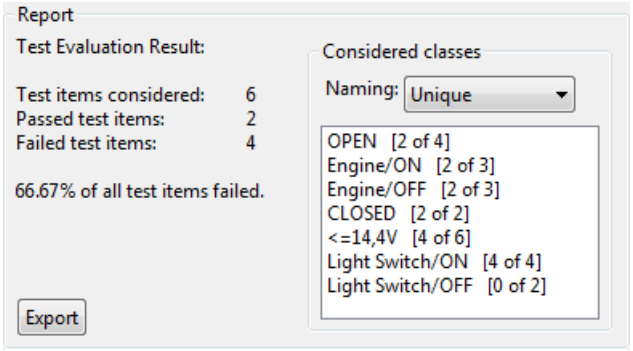

Fig. 8. Test evaluation report in CTE showing the considered test items and their failure rate

In the example given in Fig. 8, six test cases were considered for evaluation: Two passed while four failed. For all failed test cases, detailed results are given in the right part of the Figure, e.g. whenever *OPEN* was used in a test case, it failed two of four times. The last two entries *Light Switch/ON* and *Light Switch/OFF* are most interesting here: Whenever *Light Switch* was *ON*, the test case failed while with *Light Switch ON* there were no problems at all.

VII. Related Work

There is a broad body of work on combinatorial (interaction) testing [4], [16]–[25].

Elbaum et al. provide good overviews of existing prioritization approaches [26]. There has been some work on test case prioritization that considered limited resources [27], [28]. There are some known algorithm supporting prioritized test case generation. The first is an algorithm published in [29], which is an extension to [30]. For efficiency reasons, this algorithm does not consider constraints. The other approach supports constraints and is based on Binary Decision Diagram (BDD) [31]. CTE XL Professional also uses the latter [5], [12]. Search based solutions for test case generations have been presented for both, conventional [8], [32] and prioritized [33] test case generation.

A body of work on the application of the classification tree method and CTE can be found in recent work [34]–[37].

An introduction to root cause analysis can be found in [38], while there are several case studies available, e.g. [39]. The idea of combining the field of (combinatorial) testing with (root) cause analysis is not new, e.g. Cleve and Zeller try to find failure causes through automated testing [40]. The reduction of test suites down to failure-causing combinations is the background of [41], [42], so this approach might not fit too well for regression testing, where new errors can occur. The adaptation of combinatorial testing has also been discussed more recently in [43]. Beside widely positive works, Ramler et al. see limitations with the integration of cause analysis to combinatorial testing, although there is customer demand [44].

VIII. Conclusion

Regardless of the system under test, level of integration, test phase and domain, the classification tree editor is universally applicable [34]–[37]. This method has found a worldwide acceptance over the last two decades and has been used by commercial data processing, aviation and aerospace industries, and many others, for example, for examining the Hubble Space Telescope [45].

The systematic approach, starting from functional requirements, with understandable, reliable (intermediate) results, supported by an efficient, automatic test case generation, ensures that there are no gaps in the testing process and the resulting specifications.

Future work is twofold, first to ease the creation of classification trees, e.g. by importing logs [46] and second the transformation of test specification from the CTE to executable test cases.

Acknowledgment

This work is partly supported by EU grant ICT-257574 (FITTEST).

References

[1] M. Grochtmann and K. Grimm, "Classification trees for partition testing," *Softw. Test., Verif. Reliab.*, vol. 3, no. 2, pp. 63–82, 1993. [Online]. Available: http://dx.doi.org/10.1002/stvr.4370030203

[2] M. Grochtmann and J. Wegener, "Test case design using classification trees and the classification-tree editor CTE," in *Proceedings of the 8th International Software Quality Week*, San Francisco, USA, May 1995. [Online]. Available: http://www.systematic-testing.com/documents/qualityweek1995_1.pdf

[3] G. J. Myers, *The Art of Software Testing*. John Wiley & Sons, 1979.

[4] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, pp. 11:1–11:29, February 2011. [Online]. Available: http://dx.doi.org/10.1145/1883612.1883618

[5] P. M. Kruse and M. Luniak, "Automated test case generation using classification trees," *Software Quality Professional*, vol. 13(1), pp. 4–12, 2010.

[6] J. Wegener and U. Herold, "Requirements and Test Case Tracing," in *Embedded Real Time Software and Systems 2012 ($ERTS^2$)*, 2012.

[7] P. M. Kruse, J. Nasarek, and N. Condori Fernandez, "Systematic Testing of Web Applications with the Classification Tree Method," in *XVII Iberoamerican Conference on Software Engineering (CIbSE 2014)*, 2014.

[8] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *ISSTA '07: Proceedings of the 2007 international symposium on Software testing and analysis*. New York, NY, USA: ACM, 2007, pp. 129–139. [Online]. Available: http://dx.doi.org/10.1145/1273463.1273482

[9] E. Lehmann and J. Wegener, "Test case design by means of the CTE XL," *Proceedings of the 8th European International Conference on Software Testing, Analysis and Review (EuroSTAR 2000), Kopenhagen, Denmark, December*, 2000. [Online]. Available: http://www.systematic-testing.com/documents/eurostar2000.pdf

[10] P. M. Kruse, J. Bauer, and J. Wegener, "Numerical constraints for combinatorial interaction testing," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 758–763. [Online]. Available: http://dx.doi.org/10.1109/ICST.2012.170

[11] J. Petke, S. Yoo, M. B. Cohen, and M. Harman, "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 26–36. [Online]. Available: http://dx.doi.org/10.1145/2491411.2491436

[12] P. M. Kruse and I. Schieferdecker, "Comparison of Approaches to Prioritized Test Generation for Combinatorial Interaction Testing," in *Federated Conference on Computer Science and Information Systems (FedCSIS) 2012*, Wroclaw, Poland, 2012.

[13] P. M. Kruse and J. Wegener, "Test sequence generation from classification trees," in *Proceedings of ICST 2012 Workshops (ICSTW 2012)*, Montreal, Canada, 2012. [Online]. Available: http://dx.doi.org/10.1109/ICST.2012.139

[14] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[15] P. M. Kruse and J. Reiner, "Systematic design and automated execution of embedded system tests," in *Embedded Real Time Software and Systems ($ERTS^2$) 2014*, 2014.

[16] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, pp. 418–421, 2004. [Online]. Available: http://dx.doi.org/10.1109/TSE.2004.24

[17] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," *IEEE Transactions on Software Engineering*, vol. 23, pp. 437–444, 1997.

[18] Y. Lei, K. Tai, F. Inc, and N. Raleigh, "In-parameter-order: a test generation strategy for pairwise testing," in *Third IEEE International High-Assurance Systems Engineering Symposium, 1998. Proceedings*, 1998, pp. 254–261. [Online]. Available: http://dx.doi.org/10.1109/HASE.1998.731623

[19] S. Maity and A. Nayak, "Improved test generation algorithms for pair-wise testing," in *ISSRE*. IEEE Computer Society, 2005, pp. 235–244. [Online]. Available: http://dx.doi.org/10.1109/ISSRE.2005.23

[20] M. B. Cohen, J. Snyder, and G. Rothermel, "Testing across configurations: implications for combinatorial testing," *SIGSOFT Softw. Eng. Notes*, vol. 31, pp. 1–9, November 2006. [Online]. Available: http://dx.doi.org/10.1145/1218776.1218785

[21] 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.

[22] J. Czerwonka, "Pairwise testing in real world, practical extensions to test case generators," in *Proceedings of 24th Pacific Northwest Software Quality Conference*. Citeseer, 2006, pp. 419–430.

[23] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing: Research articles," *Softw. Test. Verif. Reliab.*, vol. 17, no. 3, pp. 159–182, 2007. [Online]. Available: http://dx.doi.org/10.1002/stvr.v17:3

[24] W. Grieskamp, X. Qu, X. Wei, N. Kicillof, and M. B. Cohen, "Interaction coverage meets path coverage by smt constraint solving," in *Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop*, ser. TESTCOM '09/FATES '09, 2009, pp. 97–112. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-05031-2_7

[25] A. Calvagna and A. Gargantini, "A formal logic approach to constrained combinatorial testing," *Journal of Automated Reasoning*, April 2010. [Online]. Available: http://dx.doi.org/10.1007/s10817-010-9171-4

[26] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002. [Online]. Available: http://dx.doi.org/10.1109/32.988497

[27] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *IEEE Transactions on Software Engineering*, vol. 36, pp. 593–617, 2010. [Online]. Available: http://dx.doi.org/10.1109/TSE.2010.58

[28] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Timeaware test suite prioritization," in *Proceedings of the 2006 international symposium on Software testing and analysis*, ser. ISSTA '06. New York, NY, USA: ACM, 2006, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1145/1146238.1146240

[29] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pair-wise coverage with seeding and constraints," *Information & Software Technology*, vol. 48, no. 10, pp. 960–970, 2006. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2006.03.004

[30] C. J. Colbourn and M. B. Cohen, "A deterministic density algorithm for pairwise interaction coverage," in *Proc. of the IASTED Intl. Conference on Software Engineering*, 2004, pp. 242–252.

[31] I. Segall, R. Tzoref-Brill, and E. Farchi, "Using binary decision diagrams for combinatorial test design," in *Proc. of the 2011 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2011. [Online]. Available: http://dx.doi.org/10.1145/2001420.2001451

[32] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "An improved meta-heuristic search for constrained interaction testing," *Search Based Software Engineering, International Symposium on*, vol. 0, pp. 13–22, 2009. [Online]. Available: http://dx.doi.org/10.1109/SSBSE.2009.25

[33] J. Ferrer, P. M. Kruse, J. F. Chicano, and E. Alba, "Evolutionary algorithm for prioritized pairwise test data generation," in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO) 2012*, Philadelphia, USA, 2012. [Online]. Available: http://dx.doi.org/10.1145/2330163.2330331

[34] E. Puoskari, T. E. J. Vos, N. Condori-Fernandez, and P. 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*. ACM, 2013, vol. 6, pp. 7–12. [Online]. Available: http://dx.doi.org/10.1145/2489280.2489287

[35] P. M. Kruse, N. Condori-Fernández, T. E. Vos, A. Bagnato, and E. Brosse, "Combinatorial testing tool learnability in an industrial environment," in *7th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2013. [Online]. Available: http://dx.doi.org/10.1109/ESEM.2013.49

[36] P. M. Kruse, O. Shehory, D. Citron, N. Condori Fernandez, T. E. J. Vos, and B. Mendelson, "Assessing the applicability of a combinatorial testing tool within an industrial environment," in *11th Workshop on Experimental Software Engineering (ESELAW 2014)*, 2014.

[37] N. Condori-Fernández, T. Vos, P. M. Kruse, E. Brosse, and A. Bagnato, "Analyzing the applicability of a combinatorial testing tool in an industrial environment," Technical report UU-CS-2014-008, Utrecht University, Tech. Rep., 2014.

[38] J. J. Rooney and L. N. V. Heuvel, "Root cause analysis for beginners," *Quality progress*, vol. 37, no. 7, pp. 45–56, 2004.

[39] M. Leszak, D. E. Perry, and D. Stoll, "A case study in root cause defect analysis," in *Proceedings of the 22nd international conference on Software engineering*. ACM, 2000, pp. 428–437. [Online]. Available: http://dx.doi.org/10.1145/337180.337232

[40] H. Cleve and A. Zeller, "Finding failure causes through automated testing," in *International workshop on automated debugging*, 2000, pp. 254–259.

[41] C. Nie and H. Leung, "The minimal failure-causing schema of combinatorial testing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 4, p. 15, 2011. [Online]. Available: http://dx.doi.org/10.1145/2000799.2000801

[42] L. S. G. Ghandehari, Y. Lei, T. Xie, R. Kuhn, and R. Kacker, "Identifying failure-inducing combinations in a combinatorial test set," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 370–379. [Online]. Available: http://dx.doi.org/10.1109/ICST.2012.117

[43] C. Nie, H. Leung, and K.-Y. Cai, "Adaptive combinatorial testing," in *Quality Software (QSIC), 2013 13th International Conference on*. IEEE, 2013, pp. 284–287. [Online]. Available: http://dx.doi.org/10.1109/QSIC.2013.22

[44] R. Ramler, T. Kopetzky, and W. Platz, "Combinatorial test design in the tosca testsuite: lessons learned and practical implications," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 569–572. [Online]. Available: http://dx.doi.org/10.1109/ICST.2012.142

[45] N. Tull, "Applications of specification-based testing in flight software development for hubble space telescope mission operations," 2005. [Online]. Available: http://terpconnect.umd.edu/~austin/ense623.d/projects05.d/NzingaTull-Final-Report.pdf

[46] P. M. Kruse, I. W. B. Prasetya, J. Hage, and A. Elyasov, "Logging to facilitate combinatorial system testing," in *Future Internet Testing*. Springer International Publishing, 2014, pp. 48–58. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07785-7_3