Adaptive Knowledge-Supported Testing: An Approach for Improving Testing Efficiency

Philipp Helle and Wladimir Schamai
Airbus Group Innovations
Hamburg, Germany
Email: {philipp.helle,wladimir.schamai}@eads.net

Abstract—**This paper introduces a new method for automatic test parameter generation that has been named adaptive knowledge-supported testing. The approach uses a combination of random testing for test parameter generation and machine learning and data mining techniques to optimize these test parameters based on the results from previous tests. The goal is to enable efficient testing of complex systems which cannot be tested exhaustively anymore due to the huge number of possible input combinations. The paper provides a description of the method and also results from the evaluation of a first proof-of-concept demonstrator that has been implemented to validate the method.**

Keywords*—*Testing, Adaptive Testing, Machine Learning.

I. Introduction

Increasing system complexity results in an increase in complexity of the test engineers' task to ensure that systems are correct. In recent industry practice, the verification phase is commonly the longest phase in system development and is the most critical to completing a product on time [1].

This raises the need for the development of new techniques and methodologies that can provide the test engineers with the means to achieve their goals quickly and with limited resources. These solutions succeed in removing much of the manual labour traditionally involved in the verification process. Tasks such as test execution and test report generation are now typically automated to a high degree. Model-based testing (MBT) is a new trend in industry that focuses on automatic test case generation [2] [3]. However, test parameter generation is often still a manual task [4].

Adaptive knowledge-supported testing is a new method for automatic test parameter generation. It combines methods from the area of machine learning and artificial intelligence, e.g., neural networks and methods from the area of data mining, e.g., data clustering and existing methods for test parameter generation, e.g., random testing.

Under the assumption that critical test points, i.e., stimulus combinations that lead to errors, typically occur in groups in the whole test parameter space, adaptive knowledge-supported testing allows to generate optimized test parameters from previously executed tests and their results.

This paper is structured as follows: Section 2 provides some background regarding testing and machine learning. Section 3 deals with related and prior work. Section 4 introduces the adaptive knowledge-supported testing method using a running example and, finally, Section 5 concludes the paper.

II. Background

A. Testing

"Testing is the process of executing a program with the intent of finding errors" [5]. Testing is a well-known discipline in the software and system engineering fields, and in recent decades many testing strategies have been developed, such as stress testing, fault injection, coverage-based testing, black-box, and white-box testing, or combinations of these.

However, it has been pointed out that "[in] general, it is impractical, often impossible, to find all the errors in a program" [5] and "every testing method (save exhaustive testing [..]) is less than perfect" [6]. Testing cannot guarantee the absence of errors, but it can help to discover their presence. The economics of testing, i.e., the balance between the testing effort and project time or resource constraints, depends on the selected testing strategy and the way test cases are designed, as well as the experience of the testers [5].

Half of all embedded systems development projects are way behind schedule and less than half of the designs meet 20% of the expectations in terms of functionality and performance according to the study in [7]. This is despite the fact that around half of the total development effort is spent on testing [7], [8]. These numbers underline the importance and desirability of reducing test effort by advances in the testing methodologies, especially considering the trend for "increase in software complexity [and] shorter innovation cycle times" [9].

B. Machine learning

According to a standard definition, "Machine learning is programming computers to optimize a performance criterion using example data or past experience." [10]. And more formally: "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [11].

Transferred to our application this translates as follows: The adaptive knowledge-supported testing learns from *past test results* (experience E) with respect to the task *test parameter generation* (task t) and performance measure *effectiveness of the generated test parameter sets in detecting errors* (performance measure p). So, the *test parameter generation*, measured by the *effectiveness of the generated test parameter sets* improves with the *number of past test results*.

There are different machine learning techniques, such as supervised, unsupervised or reinforcement learning, as well as classification methods, such as decision trees, naive Bayes classifier, support vector machines, neural networks, etc. The methods differ in learning complexity, classification accuracy and robustness, the possibility to interpret the generated results, and performance. Each of the classification methods has advantages and disadvantages. For example, decision trees may be computationally expensive because of the number of distinct nodes to be created. Support vector machines require more effort in setting up the learning and may also be computational expensive. The advantage of using naive Bayes models is the simplicity of learning. However, the underlying assumption is that all input values are independent, which is not always applicable to our setting, in which test parameters values may be interrelated. In turn, artificial neural networks are an alternative to dynamic and non-linear problems because they are not restricted in terms of normality, independence of input data etc.

Artificial Neural Networks (ANN) mimic the biological neural networks in their learning function. ANN are composed of connected neuron models. Each connection has a different weight that is adjusted throughout learning. A special type of artificial neural networks is the Probabilistic Neural Network (PNN). It is a four layers feed-forward network proposed by Specht [12]. When using the Dynamic Decay Adjustment (DDA) algorithm [13] it is possible to build the network dynamically based on the numerical training data. The output of the trained network are inferred rules that enable to predict the probability that new test data belongs to a certain target category. We used PNN (DDA) for our case study for learning from previous test runs and producing new test sets by predicting (i.e., selecting new test data based on the their highest probability).

III. Related Work

Random testing, i.e., random selection of test cases, is generally regarded as not only a simple but also an ”intuitively appealing” [14] technique amongst the black box techniques for test case generation. In random testing, test cases may be randomly chosen based on a uniform distribution or according to other distributions that are inferred from the operational profile of a unit under test (UuT). Hamlet [6] points out that the main benefits of random testing include the availability of efficient algorithms to generate test cases, and also its ability to provide reliability and statistical estimates. Using random test inputs allows many design requirements to be verified very quickly with minimal manual effort. Random tests also have the additional possible benefit of generating test cases that human test engineers would not necessarily think of [15]. Studies have shown that systematic testing methods are not much better at finding failures than random testing [16] and more recent research ”further support[s] the use of random testing in real-world software”[17].

However, random testing also has weaknesses, e.g., ”a vast number of test points are required” [6] and knowledge of the operational profile of a UuT are required to infer suitable distributions for the random number generators. Also, random testing usually does not produce all test cases that are needed to verify a design. The test engineer must evaluate the coverage results of the executed tests and determine, which cases remain to be tested, which can then either be written manually or generated by adjusting the random number generator in an attempt to steer the random test generation into the untested scenarios.

Several new approaches try to combine random testing with a more systematic approach to get the best of both worlds: automatic and quick test case generation coupled with a system to steer the test case generation.

Adaptive Random Testing (ART) is a method based on random testing that seeks to distribute test cases more evenly within the input space [18]. It uses two separate sets of test cases, the executed set and the candidate set, which is a set of randomly generated test points. At each iteration one or more test points are selected from the candidate set and used for a test. The criterion for selection is maximum distance from previously executed tests, which results in a more even spread of test cases in the test space. The distance function needs to be defined for each type of test. The example in the paper uses the Euclidean distance. Adaptive knowledge-supported testing also distinguishes between the executed set of tests and the candidate set of tests, which is generated using random methods. Instead of maximising the distance between test cases to evenly spread the tests we use machine learning to focus testing.

Coverage Directed Test Generation (CDG) uses coverage measurement together with a random test generator in order to assess the progress of the testing process [15]. The coverage analysis allows to modify the directives for the test generators and thereby to target areas of the UuT that are not covered well.

More recently, effort has been made to couple CDG with machine learning techniques to close the manual feedback loop from coverage analysis to test parameter generation. Machine learning, i.e., a Bayesian network, is used to observe the impact of input stimulus changes on coverage goals and a subsequent automatic steering of the input generation parameters so that the coverage is maximised [15]. It has been shown that this kind of CDG can successfully generate test directives from an analysis of observed test coverage gaps to guide the testing to completion more quickly [19][20].

CDG requires a detailed insight into the UuT to allow measuring the coverage achieved by testing, which is not easily possible in blackbox testing. In contrast to CDG, which tries to optimize coverage of the UuT, adaptive knowledge-supported testing aims at optimizing the chance to discover errors in the UuT while minimizing the required number of test runs.

IV. Adaptive knowledge-supported testing

A. Running example

The adaptive knowledge-supported testing method was developed using representative example data provided by a simulation model. The test considered was a power interrupt test, which tests the robustness of the unit under test by applying a number of power interrupts. The test parameters that typically characterize the power interrupt test and their value ranges (positive integers) are provided by Table I:

These test parameters are defined by a test engineer based on his knowledge of the UuT and the goal of the test.

Test parameter	*Min value*	*Max value*	*Smallest step size*
Number of interrupts	1	10	1
Interrupt duration (/10ms)	1	20	1
On time duration (s)	1	20	1

TABLE I: TEST PARAMETERS FOR POWER INTERRUPT TEST

Combinatorics dictates that there exist 4000 possible test points using full factorial parameter combination. We use a monitor-based testing approach described in [21] for evaluating a test run. Other automated test verdict generation approaches could be used as well. The test verdict returned by running one set of test parameters can take three distinct values:

- 0: test passed successfully
- 1: warning point, e.g., some values are anomalous but are still within the allowed value range
- 2: error point, a requirement has been violated

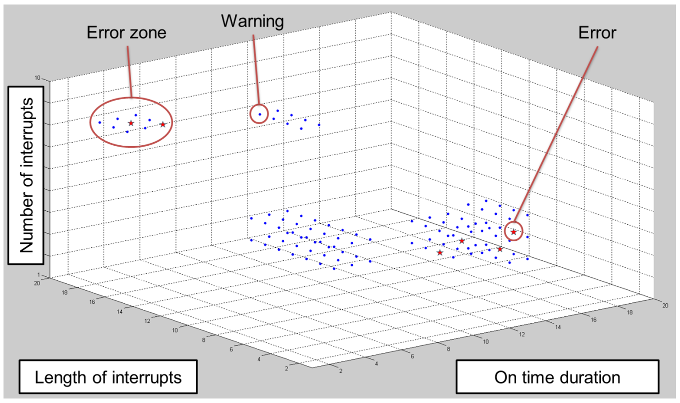

Fig. 1: Running example error distribution

For testing the adaptive knowledge-supported testing method, four error zones have been included in the total test space of 4000 test points, with 6 error points and 96 warning points as Figure 1 shows.

B. Assumptions

One general assumptions is underlying the adaptive knowledge-supported testing approach:

- Warnings and errors occur in groups in the test space: This stems from the observation that errors and warnings do not occur in an isolated fashion in the test space but, since moving from one test point to a neighbouring one represents only a very slight change of inputs, they are found in groups.

Another assumption has been made that is motivated by the running example and is depending on the test evaluation criteria and the possible values of the test verdict.

- Errors are surrounded by warnings: This represents the knowledge from past testing campaigns that testers do not stumble upon errors out of the blue but that, when the stimuli are closing in on an error then a system starts to behave anomalously but still within the bounds of the allowed, e.g., a variable value that has an upper limit starts moving towards this threshold or a variable that should be steady starts to flutter slightly but does not violate the fixed boundaries yet.

Both of these assumptions are based on lessons learnt from past test campaigns and have been confirmed internally by test experts.

C. Process

The starting point for adaptive knowledge-supported testing is always the definition of a test case that is parametrized and of all possible stimuli for the system under test for the given test case. This is done by the test engineer.

For our running example, the abstract parametrized test case can be informally described as follows:

1) Turn on the UuT.
2) After `$OnTimeDuration` start the first power interrupt by turning off the power supply and turning it back on after `$InterruptDuration`*10 ms.
3) If `$NumberOfInterrupts` is greater 1, after `$InterruptDuration`*10 ms initiate the next power interrupt until `$NumberOfInterrupts` interrupts have been executed.
4) Five seconds after the last power interrupt check if the UuT is in the normal operating mode and has successfully passed the initialisation.

The unique benefits of adaptive knowledge-supported testing come to fruition when existing test results are available. If this is not the case then adaptive testing largely corresponds to the underlying test method that is used for the test data generation, i.e., without existing rest results adaptive testing using random data generators becomes random testing.

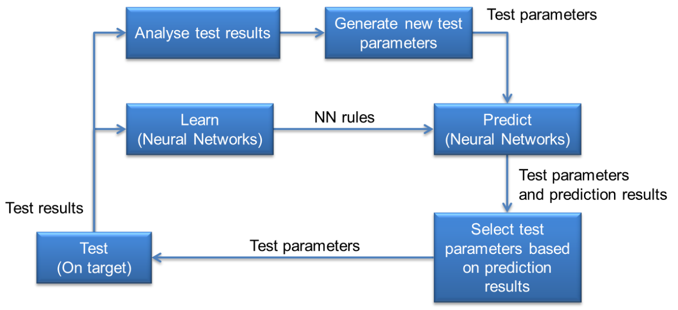
Analyse test results
Generate new test parameters
Test parameters
Learn (Neural Networks)
NN rules
Predict (Neural Networks)
Test parameters and prediction results
Test results
Test (On target)
Test parameters
Select test parameters based on prediction results

Fig. 2: Adaptive knowledge-supported testing process

Figure 2 provides an overview of the general process for the adaptive knowledge-supported testing.

1) **Analyse test results** The goal is to find starting points for the generation of new test parameter values. This may be achieved using different methods:
   - Clustering: test results are grouped into clusters. A possible starting point can then be the center of a cluster.
   - Error point: new starting points are all the discovered error points

2) **Generate new test parameters** Based on the results of the test results analysis and the derived starting points, new potential test parameters are generated to form the candidate set. Various methods can be used for that:
    - Random: new values are generated randomly
    - Stochastics: new values are generated based on stochastic distributions (e.g., Gaussian distribution)
    - Fixed step sizes: new values are generated using steps with predefined step sizes from the starting point(s)

    Other methods for test parameter generation, such as the Category-Partition Method [22] or other structured parameter generation methods could be used as well here.
3) **Learn and predict** Test results from previously executed tests are put into a neural network to derive a decision function. This function can then be used to automatically classify new input data into the three classes that are relevant for our need: test passed successfully, warning and error. New input data means test data that has not been used for learning before. Simply put, the trained neural network is used to predict which of the newly generated test input combinations from the candidate set are highly likely to produce an error or a warning. This information is used in the next step to select a desired set of new test stimuli.
4) **Select test parameters based on prediction results** Usually, test parameter generation results in a very large number of new test parameter combinations, especially when the new individual test parameter values are combined using the Cartesian product to generate new test points. Since it is not always possible to run thousands of tests, this process steps allows reducing the final number of new test points. In this step, some of the test points from the candidate set are selected for the next test execution. This can be done using different selection criteria:
    - Only test points with high error probability (according to the prediction)
    - Mix of test points with high and low error probability
    - Absolute limit for number of test points
5) **Execute test** One by one all the newly selected test points are used to drive one test and obtain a test result. Ideally, this task is automated but depending on the type of task it may also be conducted completely manually. Test execution is not in the focus of this work but has a strong impact on the number of tests that can be conveniently executed. A manual test that lasts one or two hours cannot be conducted a 1000 times with different parameters while a fully automatic test that executes in a couple of seconds can.

Note, that the process is iterative. In each iteration loop, the focus of the test effort is adapted and shifted according to the knowledge gained from the accumulated past results, hence the name of the approach.

Past test results are not necessarily limited to tests on the same version of or indeed the very same type of UuT. The power interrupt test is a very general test, that is applicable to all kinds of components. Different components with a similar start-up routine might exhibit similar failures especially if there are other influencing factors, e.g., in our case two different components might be from the same supplier and therefore use the same kind of power converters, which have a heavy influence on the behaviour reacting to power interrupts or two components might have the same kind of interface, e.g., a CAN bus interface, that is implemented using the same commercial-of-the-shelf interface controller.

Deciding if past test results from another test campaign are suitable for the current UuT is a task left to the test engineer but might, in the future, be supported by a classification of different components, e.g., using an ontology database.

D. Implementation

The adaptive knowledge-supported testing approach has been implemented in a proof-of-concept demonstrator. This demonstrator is based on the Konstanz Information Miner (KNIME) [23] tool, which is available under the GPL GNU Public License, Version 3, an open source license. KNIME is a data analytics platform for data access, transformation, mining and visualisation. It provides a basic set of data processing operations, called nodes, that can be combined graphically in a so-called workflow to achieve complex information manipulation processes.

The test bench and the UuT were implemented as a single simulation, basically a lookup table that accepts a test parameter set at a time and provides the ”test result” as a three-valued integer output as explained before. This allowed us to define the errors zones so that we could benchmark the performance of the adaptive knowledge-supported testing approach. Comma Separated Values (CSV) was chosen as the data exchange format between KNIME and the simulated test bench because it is natively supported by KNIME and an easily adaptable format that can be read in a standard editor, which eases debugging. The complete demonstrator setup is shown by Figure 3.

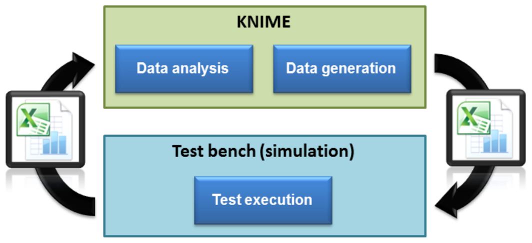

Fig. 3: Adaptive knowledge-supported testing demonstrator

E. Evaluation results

Table II shows the result from the application of the adaptive knowledge-supported testing approach to the power interrupt test. Four iterations of 100 test sets each where conducted. For comparison, we included a brute-force approach, which simple runs all 4000 possible test points and unsurprisingly discovers all warnings and errors but also has a high cost attached to it, i.e., the number of tests per uncovered error or warning is significantly higher than using the adaptive testing approach.

	Method		
Metric	*Brute-force*	*Adaptive testing I*	*Adaptive testing II*
Tested points	4000	400	400
Warnings found	96	22	67
Errors found	6	0	2
Tests/Warnings	41,7	18,2	6
Tests/Errors	666,7	n/a	200

TABLE II: COMPARISON BETWEEN BRUTE-FORCE AND ADAPTIVE TESTING

As already discussed in Section IV-C, the process step ”Select test parameters based on prediction results” permits different options for the selection of the new test parameter sets from the pool of generated test parameters. To understand these options, it is important to understand the output of the predictor. For each test set the predictor has four different outputs:

- **0**: The predicted likelihood between 0 and 1 that this test set will return a 0 result (test passed successfully)
- **1**: The predicted likelihood between 0 and 1 that this test set will return a 1 result (warning)
- **2**: The predicted likelihood between 0 and 1 that this test set will return a 2 result (error)
- **Winner**: Either 0, 1 or 2; the most likely, i.e., the one with the highest likelihood, of the three possible results.

We use the Winner value, as well as the 0 likelihood value for selecting test parameters for the next iteration of testing from the candidate set of test parameter values that were created in the ”Generate new test parameters” process step.

As can be seen from Table II two different options of the adaptive knowledge-supported testing approach have been evaluated:

- **Adaptive testing I**: All the test parameters, for which the predictor predicts a warning or error result (Winner is 1 or 2) are included in the new test set. Additionally, to fill the set up to 100 new test sets per generation iteration, the test points, for which the predictor predicts a 0 test result with the lowest likelihood are included as well.
- **Adaptive testing II**: All the test parameters, for which the predictor predicts a warning or error result (Winner is 1 or 2) are included in the new test set. Additionally, to fill the set up to 100 new test sets per generation iteration, a mix of 50 percent test points, for which the predictor predicts a 0 test result with the lowest likelihood and 50 percent random test points from the remaining points are included.

As we can see from the results, the second run, which includes randomly chosen test points fares better at detecting error and warnings than the run, which focuses on the most likely negative test results. The reason for that is that by focusing only on likely error producing test points the chance to uncover new error zones in the test space is increased. Ultimately, this means that a combination of random testing and focused testing delivers the most promising results.

Table III illustrates the iterative approach of adaptive testing. The results show that adaptive testing is able to exhaustively check an error zone once it is discovered. Using the random element in the ”Select test parameters based on prediction results” process step allows to uncover further error zones.

	Iteration				
Metric	*1*	*2*	*3*	*4*	*5*
Tested points	100	200	300	400	500
Warnings found	2	24	42	67	70
Errors found	0	2	2	2	2
Tests/Warnings	50	8,3	7,1	6	7,1
Tests/Errors	n/a	100	150	200	250

TABLE III: INCREMENTAL USAGE OF ADAPTIVE TESTING

A further evaluation was done comparing the results from the adaptive testing to pure random testing. To establish a mean value for the effectiveness of random testing, Monte Carlo simulations with 10000 runs each were conducted for different numbers of randomly (uniform distribution) selected tested points. Table IV contains the results.

Metric	*Number of tested points*				
Tested points	100	200	300	400	1000
Warnings found (mean)	2,4	4,83	7,19	9,59	24
Errors found (mean)	0,15	0,29	0,45	0,60	1,5
Tests/Warnings	41,7	41,4	41,72	41,71	41,7
Tests/Errors	666,7	689,7	666,7	666,67	666,7

TABLE IV: RANDOM TESTING RESULTS

The first evaluation results look promising. The first iteration of adaptive testing corresponds to random testing as no test results where available to optimise the test parameter generation. After the first iteration, adaptive testing proved to be more effective in detecting warnings and errors than pure random testing. It should be noted, however, that a more thorough evaluation is called for, where especially different error distributions will be evaluated and more runs of the adaptive testing approach need to be conducted to form a more substantiated statement about the overall effectiveness of the adaptive testing approach. One outcome of this work will also be a guideline for the user which supports the selection of the various possible options that our method has. Furthermore, it is planned to evaluate the approach using a real unit under test to show its usefulness in an industrial context.

V. CONCLUSION

In this paper, we present a new approach for testing models or systems. The approach leverages knowledge captured in previous tests in order to minimise the number of required tests for detecting errors. Our first case study shows that this approach is promising because of its ability to locate errors by using a fraction of the number of tests compared to a brute force or random testing approach. This is achieved by a mixture of random samples, which enable discovering new error zones and test points that focus on zones as soon as first error indicators are found.

Our case study shows the first promising results. However, the validity of the approach beyond the presented results is still an open question. It is subject to our future work to experiment with other classification methods, such as support vector machines or naive Bayes classifiers, as well as, evaluating the approach using different sets of randomly generated training data. Moreover, we plan to automate the workflows in KNIME in order to minimize the manual effort for setting up and running the tests.

In addition to that, we plan to run a more thorough evaluation campaign. In this we will use a larger example case with a higher number of possible test input combinations to compare our approach to a number of systematic testing approaches in terms of complexity, runtime and testing efficiency.

REFERENCES

[1] R. S. Pressman, Software Engineering - A Practitioner's Approach, 8th Edition. McGraw-Hill, 2014.

[2] J. Peleska, "Industrial-Strength Model-Based Testing - State of the Art and Current Challenges," ArXiv e-prints, Mar. 2013.

[3] M. Shafique and Y. Labiche, "A systematic review of model based testing tool support," Software Quality Engineering Laboratory, Department of Systems and Computer Engineering, Carleton University, 2010, p. 21.

[4] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," Software Testing, Verification and Reliability, vol. 22, no. 5, 2012, pp. 297–312.

[5] G. J. Myers, C. Sandler, and T. Badgett, The art of software testing. John Wiley & Sons, 2011.

[6] R. Hamlet, "Random testing," Encyclopedia of Software Engineering, 1994.

[7] V. Encontre, "Testing embedded systems: Do you have the guts for it," IBM, November, 2003.

[8] A. Helmerich, N. Koch, L. Mandel, P. Braun, P. Dornbusch, A. Gruler, P. Keil, R. Leisibach, J. Romberg, B. Schätz et al., "Study of worldwide trends and r&d programmes in embedded systems in view of maximising the impact of a technology platform in the area," Final Report for the European Comission, Brussels, Belgium, 2005.

[9] P. Liggesmeyer and M. Trapp, "Trends in embedded software engineering," IEEE Software, vol. 26, no. 3, 2009, pp. 19–25.

[10] E. Alpaydin, Introduction to machine learning. MIT press, 2004.

[11] T. M. Mitchell, Machine learning. McGraw-Hill, 1997.

[12] D. F. Specht, "Probabilistic neural networks," Neural Networks, vol. 3, no. 1, 1990, pp. 109–118.

[13] M. R. Berthold and J. Diamond, "Constructive training of probabilistic neural networks," Neurocomputing, vol. 19, no. 1, 1998, pp. 167–183.

[14] L. J. White, "Software testing and verification," Advances in Computers, vol. 26, no. 1, 1987, pp. 335–390.

[15] J. S. Vance, "Application of bayesian networks to coverage directed test generation for the verification of digital hardware designs," Ph.D. dissertation, University of Pittsburgh, 2010.

[16] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," Software Engineering, IEEE Transactions on, no. 4, 1984, pp. 438–444.

[17] A. Arcuri, M. Z. Iqbal, and L. Briand, "Random testing: Theoretical results and practical implications," Software Engineering, IEEE Transactions on, vol. 38, no. 2, 2012, pp. 258–277.

[18] T. Y. Chen, H. Leung, and I. Mak, "Adaptive random testing," in Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making. Springer, 2005, pp. 320–329.

[19] S. Fine, A. Freund, I. Jaeger, Y. Mansour, Y. Naveh, and A. Ziv, "Harnessing machine learning to improve the success rate of stimuli generation," vol. 55, no. 11. IEEE Transactions on Computers, 2006, pp. 1344–1355.

[20] S. Fine and A. Ziv, "Coverage directed test generation for functional verification using bayesian networks," in Proceedings Design Automation Conference. IEEE, 2003, pp. 286–291.

[21] P. Helle and W. Schamai, "Towards an integrated methodology for the development and testing of complex systems," in VALID 2013, The Fifth International Conference on Advances in System Testing and Validation Lifecycle, 2013, pp. 55–60.

[22] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," Communications of the ACM, vol. 31, no. 6, 1988, pp. 676–686.

[23] M. R. Berthold et al., "KNIME: The Konstanz Information Miner," in Data Analysis, Machine Learning and Applications, ser. Studies in Classification, Data Analysis, and Knowledge Organization, C. Preisach, H. Burkhardt, L. Schmidt-Thieme, and R. Decker, Eds. Springer Berlin Heidelberg, 2008, pp. 319–326.