Boosting Exploratory Testing of Industrial Automation Systems with AI

Raphael Eidenbenz*, Carsten Franke†, Thanikesavan Sivanthi‡ and Sandro Schoenborn‡
*Hitachi ABB Power Grids Research, Switzerland
raphael.eidenbenz@hitachi-powergrids.com
†Munich University of Applied Sciences, Germany
carsten.franke@hm.edu
‡ABB Research Center, Switzerland
thanikesavan.sivanthi@ch.abb.com, sandro.schoenborn@ch.abb.com

***Abstract*—Testing of large and complex industrial control systems is challenging as the space of possible input and environmental parameters is large. Searching the entire space for potential failures is practically infeasible. This paper introduces an industrial control system robustness testing problem and evaluates artificial intelligence (AI) based strategies to efficiently explore the space and to identify parameter sets that can cause the system to fail. The proposed solution approach uses regression techniques to speed up the search and clustering methods to identify parameter sets that represent distinct system failures.**

I. Introduction

The development of control systems and corresponding control algorithms for large industrial applications is a complex and arduous task. Such systems are often mission or safety critical. Consequently, the functional correctness and the robustness of the resulting systems are of utmost importance. Thus continuous integration (CI) [5] testing of the control logic under development is essential in order to identify potential problems early enough in the development cycle. The testing of control logic of large industrial applications is complex due to the high-dimensional input and output parameter space. The inputs of the control logic are subject to external boundary conditions which are usually provided by the end customers who detail the specific input ranges under which the industrial control system will operate. The resulting control system must operate correctly when the inputs are within their stipulated ranges.

Two widely used testing approaches in industry are scripted testing and exploratory testing. In scripted testing the tester executes a known set of test cases and test steps, while in exploratory testing the tester interactively runs tests, learns, and creates new tests based on the knowledge gained from the tests done so far. The advantage of exploratory testing is that by taking the history into account, the tester can focus on more promising test cases and find issues or operational limits of a system. Exploratory testing is also recently used in the context of agile testing [13]. Scripted testing is well suited for automation, but the tests are limited by the script with little room for exploration. On the other hand, the efficacy of the exploratory testing is highly dependent on the experience and skills of the tester [8]. Alternatively, combinatorial testing in which all t-way combinations of parameter values are covered can be used, but requires long testing times [10]. In a recent study [6], it is also shown that exploratory testing is the preferred testing technique among software testing practitioners. Of late, techniques based on Artificial Intelligence (AI) have been considered for test exploration. [7] presents the state of the art of applying machine learning (ML) and AI in software testing. [1] uses a combination of multi-objective search and neural networks to guide and to prune solutions from the search space. [2] uses Neural Networks for modeling the system under test and backward reasoning for identifying new test cases. Both approaches are based on supervised learning using labeled data. Most often in industrial settings the labeled data are expensive and time-consuming to obtain. However, simulation models for systems under test are mostly common. Another possibility for exploratory testing is the application of active learning. Active learning [12] is a machine learning algorithm that is applied when a solution space is large and the evaluation of individual solutions expensive. By probing some solutions active learning iteratively builds a model in the background that can predict the characteristics of individual solutions. In order to generate this prediction model, active learning strategically decides which solution instances to test next based on the history and the current model. The selection of the next instance to test differs based on the chosen sampling approach, but in general tries to minimize the overall number of test executions. As such, active learning iteratively trains a regression function that maps system inputs to outputs with limited data by querying the data instances to be labeled. [11] uses active learning based on Gaussian Process regression for safe exploration where critical and unsafe data instances are avoided. The Gaussian Process regression is a flexible method to deal with nonlinear regression problems. It calculates the probability distribution over all admissible functions that fit the data.

This paper describes an AI based solution for exploratory

The work was conducted while Carsten Franke was with Hitachi ABB Power Grids Research Switzerland.

The authors would like to thank Michael Schneeberger for fruitful discussions and workflow illustrations.

testing of the control logic under development. The solution helps to automatically predict inputs that can result in failures of the control logic within a CI nightly build. To this end our solution strategically samples the test parameter space by querying the actual system under test, or a high fidelity simulation thereof, and builds a regression model that is used in the subsequent steps to efficiently identify system conditions in which the industrial control system is likely to fail. In the last part of our proposed process, the identified problem cases are clustered and finally represented by one parameter set per cluster. This helps the engineers to focus on the various problem cases to adapt the underlying control system so as to guarantee stable operation under all customer specified conditions.

The remainder of the paper is structured as follows. Section II presents the industrial use case and formally defines the exploratory testing problem. Section III describes the solution approach in detail. Section IV presents the evaluation experiments and results. Section V draws conclusions and presents directions for future work.

II. Industrial Use Case

The case study considered in this work is robustness testing of industrial automation and control systems. In particular, the AI assisted exploratory testing concepts and methods have been developed for and evaluated against a use case of a power grid control system that is tested under system context in a virtual environment, i.e., in an offline, non-real-time simulation in MATLAB Simulink. The primarily studied test cases thereby simulate harsh conditions, such as short circuit situations, and test whether the control system can endure the situation without ceasing normal operation.

Throughout this paper, we present results that refer to such a robustness testing of a power grid control system. The respective test case takes 13 parameters as input, of which 11 are continuous and 2 are categorical. The parameter values determine the power grid control system environment and operation modes of the simulation. The simulation output values of interest are the maximum normalized values for voltage and current experienced during the simulation. If the voltage or the current exceeds a given threshold value at any time during the simulation, the test is considered to have failed, since in the real system the control would have failed to stabilize the situation. Each individual simulation run takes about 5 minutes.

A. Formal model

The considered type of test cases can be described in a formal way as follows. The test case takes n parameters $x_1, x_2, \dots, x_n = X$ as input, including both continuous and categorical values, and exhibits one continuous output signal y. Whether the test fails or succeeds is determined by applying a threshold τ on output signal y. This is, the test succeeds if $y < \tau$, and it fails otherwise. We denote the binary test outcome when applying threshold τ by $y^{\tau} \in \{0, 1\}$. Thereby $y^{\tau} = 0$, if the test succeeds, and $y^{\tau} = 1$, if it fails.

In addition to the information whether an input parameter x_i is continuous or categorical, each continuous parameter x_i is associated with a range $\Xi_i = (x_i^{min}, x_i^{max})$ of possible values, and each categorical parameter x_i is associated with a set Ξ_i of possible values.

Note that in the studied robustness test case it holds that $n = 13$, $y = \max\{\max_t \bar{I}, \max_t \bar{V}\}$, i.e. the output signal is the maximum normalized current $\bar{I}$ or voltage $\bar{V}$ experienced in any time step t of the simulation, and τ typically equals 1.0, since y is normalized a priori. Moreover, the parameter ranges (x_i^{min}, x_i^{max}) are indicated by power system domain experts and represent the ranges.

B. Exploratory Testing Problem

The general goal of exploratory testing is to find input parameter points that make the test fail, i.e., find $x_1, x_2, \dots, x_n$ so that

$$x_i \in \Xi_i \quad \forall i \in \{1, \dots, n\}, \text{ and}$$

$$y^{\tau}(x_1, x_2, \dots, x_n) = 1.$$

On the basis of these failing parameter points, the control system engineers can then investigate the reasons for the failures and improve the control system.

To establish automated exploratory testing that is useful in practice, some additional constraints must be met:

- **Limited Number of Failure Points.** The number of failing parameter points that are presented to the engineer must be limited to a number that an engineer can investigate in a reasonable amount of time. In our case, we would not like to present more than 10-15 points. For efficiency it is thus crucial for the automated exploratory testing to find failure points that have different root causes for failing, so that, in the best case, the engineer can obliterate a different issue with each investigation of a failure point. Based on the assumption that failing points that are close in the parameter space have a similar failure root cause, it is desirable to find failing parameter points that are not too close.
- **Limited Execution Time.** The overall search procedure should conclude within 12 - 60 hours so that it can run over night or during the weekend. Since the execution of the simulations underlying an individual test is compute intensive, a straightforward enumeration based exploration of the parameter space is not feasible in the given time. A more efficient exploration procedure is needed. Such an exploration method might also choose the sequence of points to be sampled strategically.
  For our particular use case the number of tests that we can afford to execute is about 700 for the search during the weekend. There is an additional benefit, if the search procedure needs only about 150 test executions, since it can be done overnight.

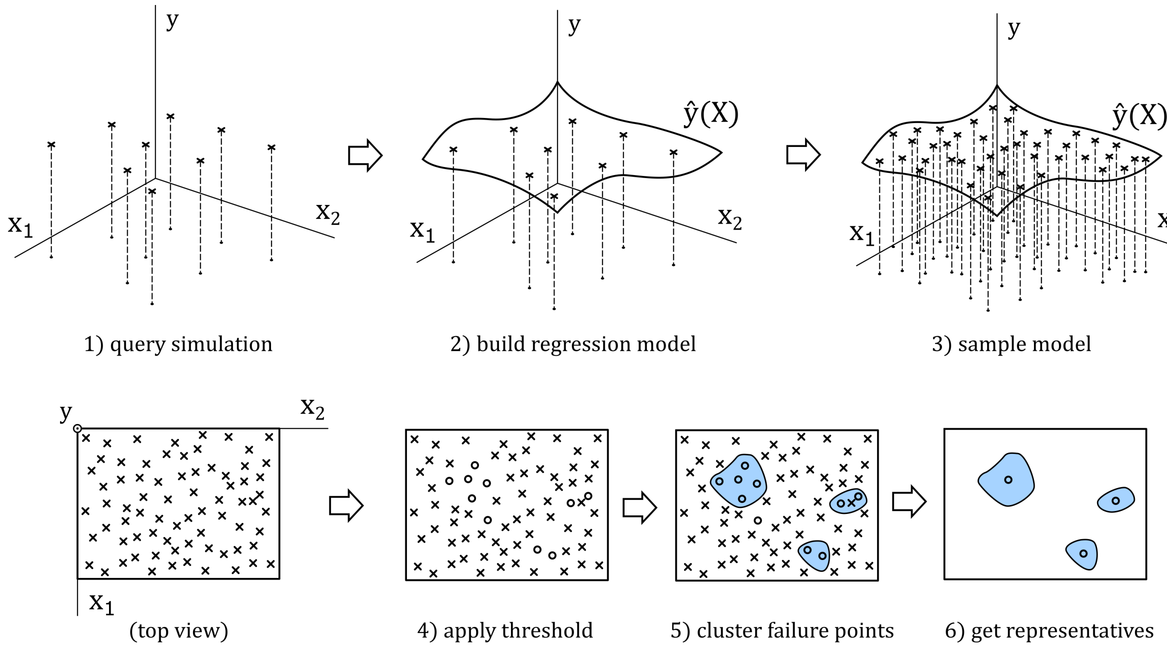


Fig. 1. The six steps of the automated exploratory testing solution approach.

III. Solution Approach

The proposed solution approach addresses the exploratory testing problem described in Section II. It provides an efficient search of failure regions in the parameter space by the following sequence of steps, which is illustrated in Figure 1.

1) In the first step, the parameter space $\Xi = \Xi_1 \times \ldots \times \Xi_n$ is queried at a number of strategically chosen points $X_1, \ldots\ X_m \in \Xi$ by running the simulation with the respective parameters. Herewith, we learn the output signals $y_1, \ldots, y_m$ at the points $X_1, \ldots\ X_m$.
2) Then a regression model $\hat{y}$ is fitted to the queried points X_i and output signals y_i. This model can now estimate the output signal y for a given point $X \in \Xi$ efficiently, which we denote as $\hat{y}(X)$.
3) In the third step, the parameter space is now explored on a larger scale by querying the efficient approximation $\hat{y}$.
4) The potentially large number of failing parameter points $F \in \Xi$ is identified by applying the threshold τ to the approximate output signals $\hat{y}$.
5) In the fifth step, the failure points F are clustered into different interesting regions $C_1, \ldots, C_k$.
6) Finally, one representative point X_{C_i} of each cluster is selected and reported to the engineers for a detailed root cause analysis.

A more detailed discussion of each step is given in the following.

A. Simulation Querying and Regression Model Training

Steps 1 and 2 train a regression model that predicts the behavior of the control system with respect to the relevant output signal y, which in our use case is the maximum scaled voltage or current experienced during a simulation. There is a plethora of regression methods that could be used for this task in general. What narrows down the choice of promising methods for the exploratory testing problem, is the fact that, unlike most common regression settings, the data is not given a-priori in our use case. Instead, we can produce the data in an online fashion, i.e., we can choose a parameter point X or a set of parameter points and query the test simulation to receive $y(X)$. Moreover, the given constraints of limited execution time require that we choose methods that are particularly good if the number of given data points is low. Hence, methods are most promising if they include a strategy for iteratively choosing the points to be sampled in such a way that the regression accuracy is maximized, already so for low numbers of points. This is in contrast to most regression settings, where the data is typically given a-priori, and the regression is computed offline.

1) Active Learning: One family of methods that includes a strategic choice of which point to query next is called *active learning*. With active learning, the regression model is built step by step. In each step the test is queried at a point for which the current prediction model experiences highest uncertainty, and the model is further trained with this latest sample. Such active learning works with essentially any machine learning algorithm that can provide an uncertainty measure.

In particular, we consider *Gaussian process regression (GPR)*, which takes a particular Bayesian approach to regression in that it learns, rather than one concrete function, a probability distribution over all possible functions in a large family of functions where the probability of a function reflects

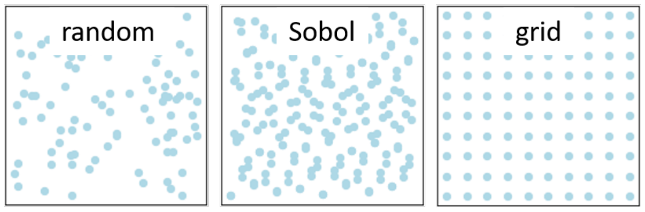


Fig. 2. Random vs. Sobol sampling vs. grid

the estimated fit to the data. For a given point X a GPR model can provide $\hat{y}(X)$, the output value of the most likely function at X, as well as a measure of uncertainty such as the standard deviation at point X.

Another class of regression methods that naturally provide an uncertainty measure are *ensemble methods*. In the most popular ensemble method, *Random Forests*, an ensemble of decision trees is trained using different subsets of the training data. For a given point X the generally different predictions $\hat{y}(X)$ provided by the trees represent a probability distribution over the estimated output signal at X. The different predictions can be averaged to provide a single estimate $\hat{y}(X)$, but they can also be used to provide a measure of uncertainty at X, e.g. the standard deviation.

In our implementation of active learning, we use a *pool-based sampling* to decide which point to query next. In pool-based sampling, a random set of candidate points is chosen; the uncertainty for each candidate point is calculated using the current regression model; finally, the candidate that exhibits the highest uncertainty is selected.

2) Quasi-Monte Carlo Sampling: As contrast to active learning, we employ also a Quasi-Monte Carlo sampling strategy that does not iteratively build a model, but queries a given number of points a priori and then builds the regression model using all those points. We thereby strive to select the set of points strategically so as to optimize the accuracy of the trained regression model.

For this point selection, it is important to cover the search space as well as possible, while at the same time avoiding aliasing effects that might occur with a selection along a grid. On the other hand, choosing points uniformly at random leads to unwanted clusters and relatively large uncovered areas. A sampling method that has proven to avoid both problems and to work well in practice is sampling according to low discrepancy Sobol sequences [3]. Refer to Figure 2 for an illustration of the differences between random sampling, Sobol sampling and grid sampling. Sobol sampling works with any supervised machine learning algorithm, e.g. Gaussian process regression, Random Forests or Neural Networks, as we first select a set of points, label those points, and then train the model with the labeled points.

B. Searching for Interesting Parameter Regions

Once the regression model is trained, it is then used to search efficiently for interesting regions, i.e., regions with mostly failing points.

The method that we employ for this purpose is a large scale space exploration, where we select a large set of points and use the regression model to predict the respective output value. To avoid issues of aliasing or larger unexplored regions, we once again employ Sobol sampling also for this task.

Then the succeeding points, where $\hat{y} < \tau$, are discarded. Among the failing points, clusters are formed using standard clustering methods, e.g. k-Means or DBScan.

C. Choosing Representative Points

Since the number of failure points that should be reported to the engineer must be limited, we apply as a last step, a method to derive one representative point for each cluster that has been found in the previous step. Moreover, the chosen points should well represent the root cause of the failure clusters. For this purpose, we choose the representative points that are far the from cluster boundaries, and near to the center of gravity of the clusters.

In particular, we choose the representative points as follows for the two implemented clustering methods:

- For k-Means, we choose the point that is closest to the center of the cluster, the so called centroid.
- For DBScan, we choose one of the *core* points uniformly at random. Core points have the property that the ball with a specified radius ϵ around the point contains at least k points, where ϵ and k are DBScan parameters that determine how tightly or loosely DBScan shall create the clusters.

D. Integration into Testing Workflow

We have implemented the described exploratory testing framework in Python in such a way that it can address the test cases as described in Section II, as well as other tests that exhibit a similar structure. It allows for execution of various types of test cases, including several test cases that are written in MATLAB. The framework harnesses MATLAB Engine for Python for the purpose of instrumenting MATLAB from Python.

The exploratory testing framework is integrated into the development cycle of the control logic using the workflow as shown in Figure 3. Engineers commit their changes of the control logic to a repository from which the Jenkins Continuous Integration (CI) Server builds a new system. The Jenkins CI Server triggers the execution of the test framework for a defined time to train a regression model. This regression model generation can be limited in time to fit a weekend or a night. This depends on the specific project setup and the envisioned quality. After training the regression model, interesting regions are extracted and clustered. This step takes less than 5 min also for larger models. Based on these clusters, representative solutions are selected. Only these finally selected parameter points, together with some high level properties such as the (estimated) overall failure rate, are shown in the reports of the Jenkins CI server. The engineers inspect the simulations at those selected points one by one in more details, and adapt the

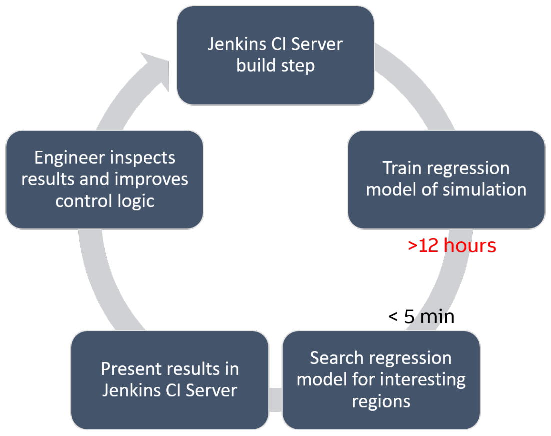

Jenkins CI Server build step
Train regression model of simulation
>12 hours
< 5 min
Search regression model for interesting regions
Present results in Jenkins CI Server
Engineer inspects results and improves control logic

Fig. 3. Automated exploratory testing workflow.

control logic and/or the test case according to their findings. Then the next iteration of the continuous integration process starts. Note that due to the changes to the control logic, the function to be learnt, y, is not the same as in the previous iteration. Thus, the regression model building is started from scratch. Depending on the quality needs, the iterative process is stopped when the failure rate is below a certain threshold. In the best case, the exploration will find no failures anymore.

IV. Evaluation Experiments and Results

In this section, we detail the experiments performed to evaluate the feasibility of our exploratory testing solution approach and to evaluate which sampling strategies and regression methods perform best. All experiment results are obtained with respect to the real robustness use-case as presented in Section II to ensure practical relevance.

A. Baseline

In order to evaluate the quality of different search and learning strategies, we first ran a long-term brute-force exploration of the robustness test case to generate a ground truth baseline of the function y. For the purpose of this baseline generation, we used one dedicated server that sampled the simulation in the bounded parameter space for several weeks to provide us with 10000 data points. For this large-scale exploration, we applied the Sobol sequence sampling method as introduced above to decide which points to sample.

We use the detailed results from the 10000 simulation runs as the ground truth and compare the various algorithms with the information extracted from this empirical data.

For the 10000 parameter points based on Sobol sequence sampling, the failed and the successfully passed tests have been identified by applying the default threshold. The finding is that the baseline exhibits a failure rate of about 13.6%, i.e., in 1363 out of the 10000 simulations the maximum voltage and current exceeded the threshold, which would indicate that the control system would fail to control the situation under the simulated environment. Figure 4 shows the distribution of the

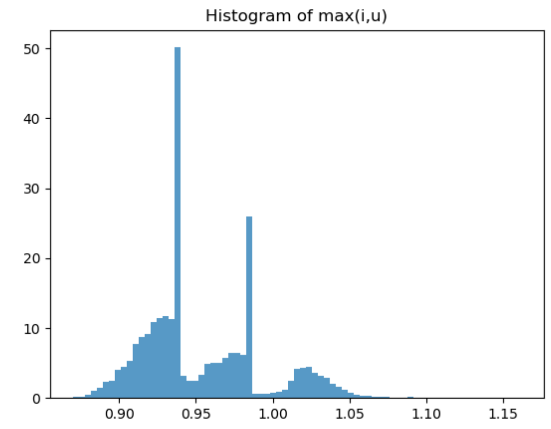


Fig. 4. Histogram of the 10000 baseline samples of y, showing the distribution.

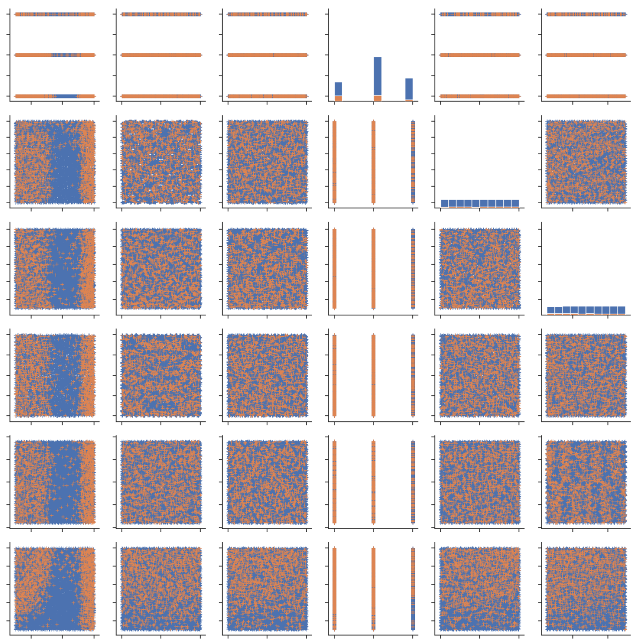


Fig. 5. Excerpt of the pairwise baseline plot. Orange points indicate failures, plue points indicate successes. Note that failures are overemphasized, because they are drawn on top of the blue points.

sampled values of y. There are two clear peaks at about 0.93 and 0.98, which are due to the configuration of the control logic. Moreover, we see the failure cluster beyond the default threshold value $\tau = 1$, which could not be contained by the control system.

In a first step, we inspected pairwise plots visually to see if there are clear failure regions. Figure 5 shows a subset of the 13x13 parameter pairwise plots. In the general, inspecting the pairwise plots revealed only very few tendential patterns. For

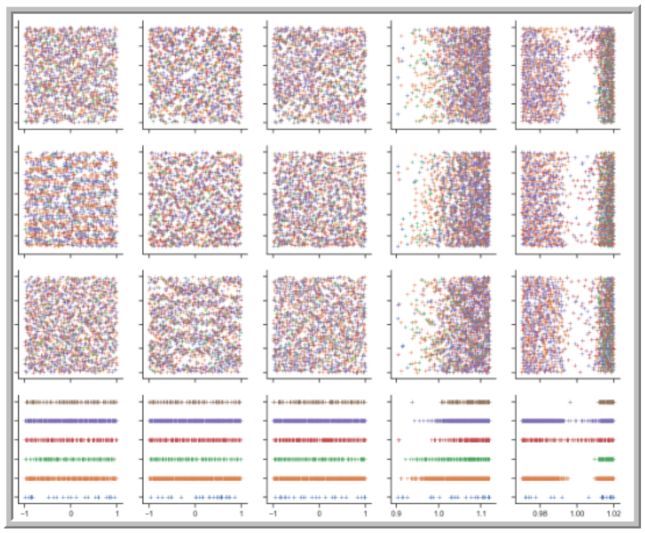


Fig. 6. Excerpt of pairwise baseline plots of failure points only. The colors represent 5 different clusters as found by DBScan.

instance, the data shows that the system is more likely to fail if the first column parameter in Figure 5 is either significantly larger or smaller than its nominal value. In general, the failure and the successfully passed tests seem well mixed, and failure regions are hard to identify by human inspection. However, this seems to be mainly due to the fact that much information is lost when embedding the 13 dimensional space into 2 dimensional spaces. Clustering algorithms find clear failure clusters in the baseline data. Refer to Figure 6 for an illustration with pairwise plots of the clusters found by DBScan in the failure points of the baseline. Each cluster is represented in the figure by one color. The last row shows the distribution of each cluster in the respective column dimension, i.e., for different parameters.

The baseline mainly serves as a ground truth for our evaluations when assessing accuracy of trained regression models $\hat{y}$, i.e., the percentage of correct classifications into failure or success. We employ it to a lesser extent when evaluating the usefulness of the chosen representative points. To judge whether each representative point indeed stems from a different root cause requires in-depth analysis by an engineer. Unfortunately, adding such manual evaluation for all sets of representative points produced during our experiments was out of scope of this research work.

B. Evaluation Implementation

For the sake of running the evaluation experiments, we implemented a testing framework in Python. The framework uses commonly available machine learning libraries, in particular SciKit-learn [9] and TensorFlow[1] with Keras[2], as well as the lesser know ModAL[3] library for active learning [4]. Moreover, it instruments Matlab using the Matlab Engine API for Python[4] to execute the Matlab Simulink simulations and fetch the results as part of the learning.

We conducted a series of experiments to compare the performance of various regression algorithms. The measure of interest is the *accuracy* in predicting whether the system will withstand the harsh conditions or not for the given environment parameters. More concretely we measure the accuracy by the rate of correct predictions, i.e., $|\{X \in \Xi \mid \hat{y}^\tau(X) = y^\tau(X)\}|/|\Xi|$, where Ξ is the parameter space. For our evaluation, we used our test set of 10000 baseline points as the ground truth Ξ. In addition to prediction accuracy, we measure precision and recall, as well as the combined measure, the F1 score, which equals $2 \cdot precision \cdot recall/(precision + recall)$. Since we expect our data to be particularly skewed, i.e., we expect the rate of failures to be much lower than the rate of successful test case executions, the F1 score likely provides the best measure for comparing performances.

The following regression algorithms have been run on the short circuit test both with active learning and with Sobol sequence sampling.

- Gaussian Process Regression with a Matérn kernel
- Decision Tree with maximum tree depth 5
- AdaBoostedTree with maximum tree depth 5
- Random Forest with maximum tree depth 10 and 100 estimators
- Random Forest with maximum tree depth 50 and 500 estimators

We ran all experiments for two different threshold levels, $\tau = 0.98$ and $\tau = 1$. This is to see whether there is a sensitivity to a variation in the threshold level, and potentially identify regression methods that make our solution more robust. In the presentation of results, we focus on our main use case, $\tau = 1$. Thus, all results refer to this case, unless mentioned otherwise.

In each experiment the regressor is trained on up to 500 points[5], since this is roughly the number of points that can be explored on a standard continuous integration server within a weekend. In each step of an experiment, the accuracy, precision, recall, and F1 score is calculated against our baseline.

C. Evaluation Results

The main results of our experiments are summarized in Figure 7. The two plots on the left side show the performance in terms of F1 score and accuracy of all studied regression algorithms when using active learning. The two plots on the right side show the F1 score and accuracy of all regression algorithms when sampling according to Sobol sequences.

Figure 8 provides more details on the precision, recall, and F1 score of Gaussian Process regression and Random Forests with maximum tree depth 50 and 500 trees. Again, the left side shows the performance with active learning; the right side shows the performance with Sobol sequence sampling. For

[1]https://www.tensorflow.org
[2]https://keras.io
[3]https://github.com/modAL-python
[4]https://www.mathworks.com/help/matlab/matlab-engine-for-python.html
[5]A few experiment series went up to 1000 points to understand whether the regression quality could be significantly improved by spending more calculation time.

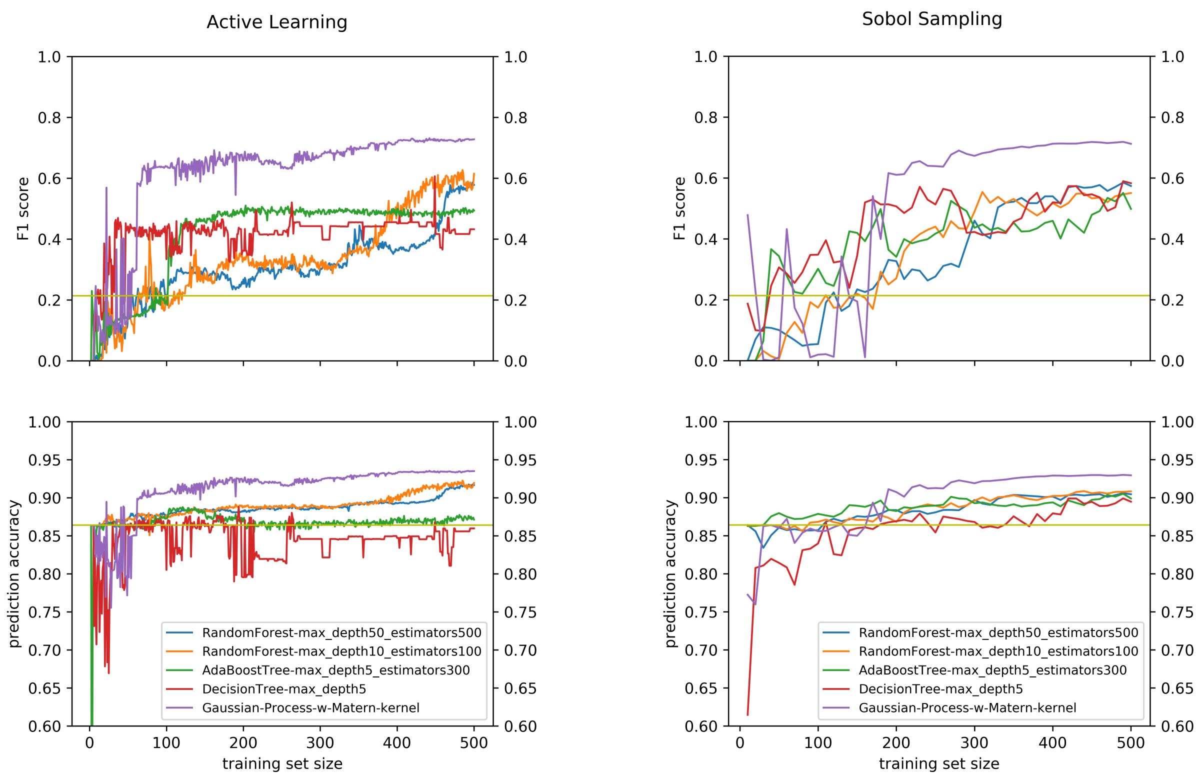


Fig. 7. Comparison of regression algorithms performance in terms of F1 score and prediction accuracy at $\tau = 1$ depending on the number of samples. Note that for active learning there is a data point at each training set size, for Sobol sampling the data comes at steps of 10.

reference, we added the expected F1 score of a trivial random predictor, which chooses from $\{0, 1\}$ uniformly at random (yellow line). This F1 score can be computed by $\rho/(\rho + 0.5)$, where $\rho = 0.136$ equals the failure rate in the baseline data.

Figure 9 shows the impact of different threshold levels on the performance discrepancies between active learning and Sobol sequence sampling by the example of Gaussian Process regression and Random Forests.

1) General Observations: Before we elaborate on more details of the results, we can state the following general insights from the experiment results:

- One main insight from the experiments is that it is generally possible to achieve an acceptable accuracy of about 90% with only 200 or 250 samples. It can be further pushed well above 90% if we can afford sampling up to 500 points.
- Overall, Gaussian Process regression scores best, with only a few exceptions.
- There are cases where active learning shows clear advantages over Sobol sampling, if one can afford less than 200 samples. In particular for Random Forests at $\tau = 0.98$ (see Figure 9, bottom right).

2) Quasi Monte Carlo Sampling versus Active Learning: The first more detailed investigation focuses on the comparison between Quasi Monte Carlo Sampling (Sobol sequences) and Active learning. As illustrated in particular by the bottom right plot of Figure 9 and to a lesser extent by the Gaussian Process regression plots in Figure 8 active learning can provide better learning up to a certain number of samples for some regression algorithms. In particular, we observed this phenomenon for Random Forests and Boosted Trees, if the time allows only to investigate up to 200 data points.

However, as Figure 7 shows, active learning does generally not have a significant advantage over Quasi Monte Carlo sampling. On the contrary, Sobol sampling performs slightly better in a majority of cases, in particular if we can use more than 250 samples. The top right example in Figure 9 shows that in some rare cases active learning can perform particularly bad, which seems to be due to some unfortunate choices in the random candidate set for pool-based sampling. In the particular example, Gaussian Process regression did not reach an accuracy above 75% after 500 samples. One explanation is that the sampling gets stuck in certain regions rather than exploring the wider space. Such a case is very unlikely with Sobol sequence sampling.

For the reasons that Sobol based Quasi Monte Carlo sampling seems more robust and that active learning shows no clear learning advantage over Sobol sampling, we conclude that Quasi Monte Carlo sampling is preferable in our use case.

3) Comparing Regression Methods: The second investigation focuses on the comparison between the various studied

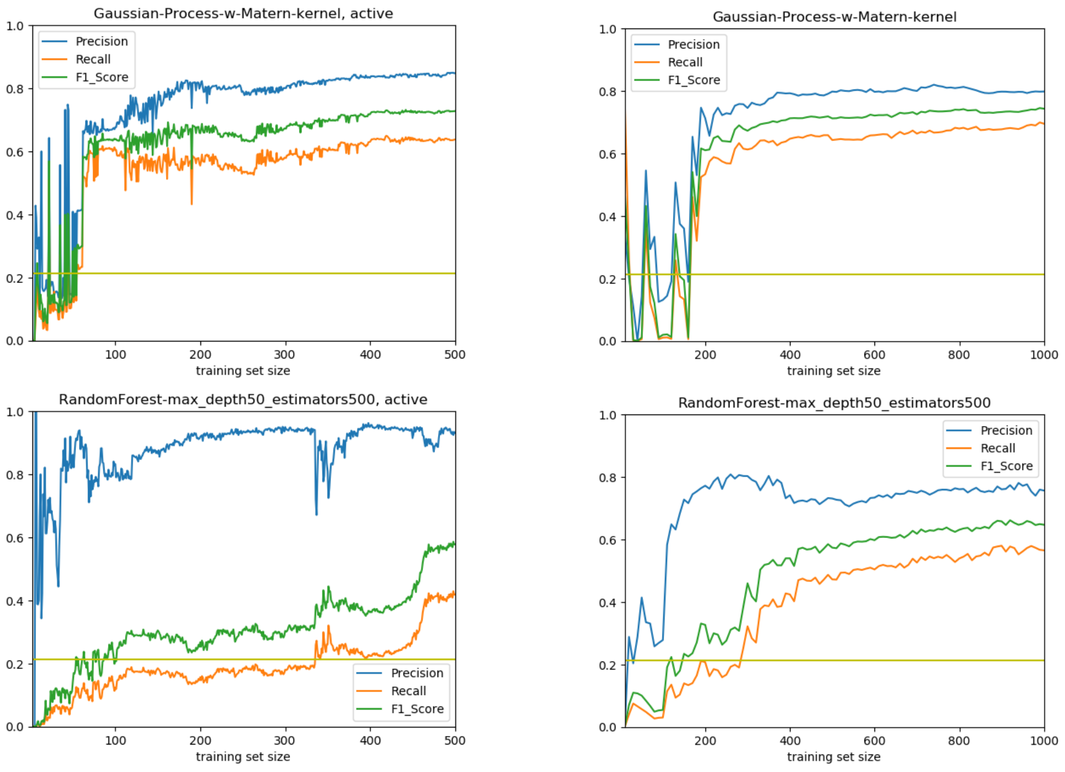


Fig. 8. Detailed comparison of Gaussian process regression and Random Forests (with depth 50) in terms of precision, recall, F1 score and prediction accuracy at $\tau = 1$ depending on the number of samples used for training. The yellow line marks the expected F1 score of a trivial random predictor.

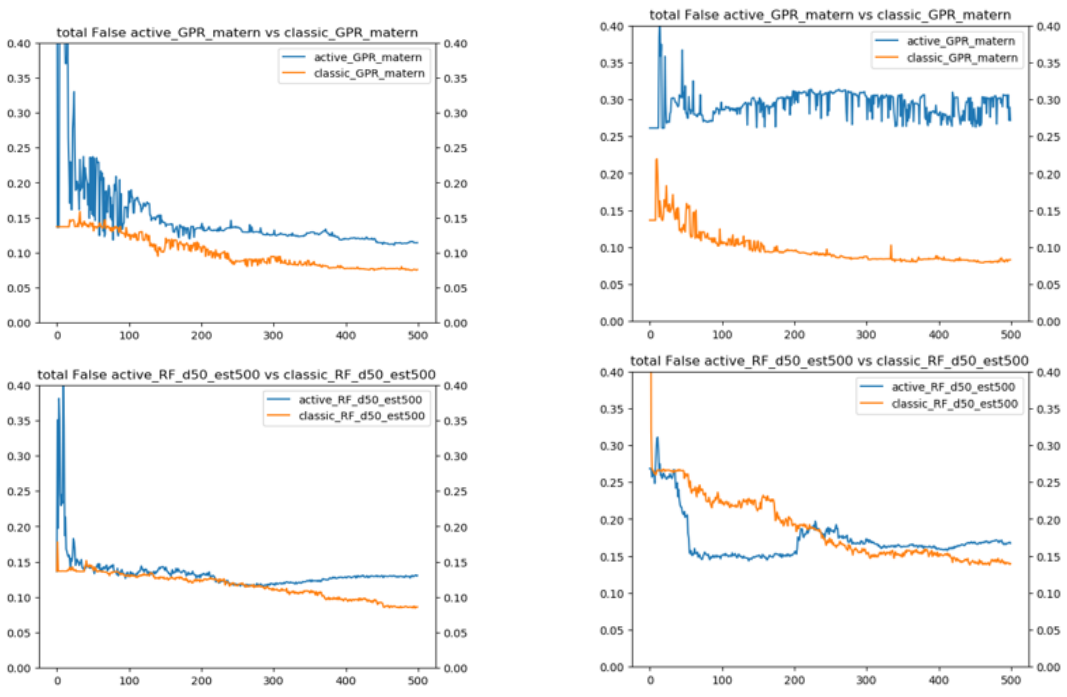
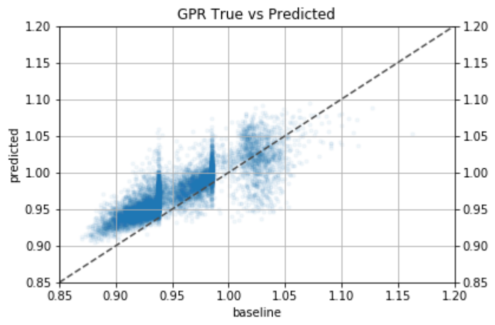


Fig. 9. Rate of false predictions of Gaussian Process regression (top) and Random Forests (bottom) with active learning at $\tau = 1$ (left) and $\tau = .98$ (right).



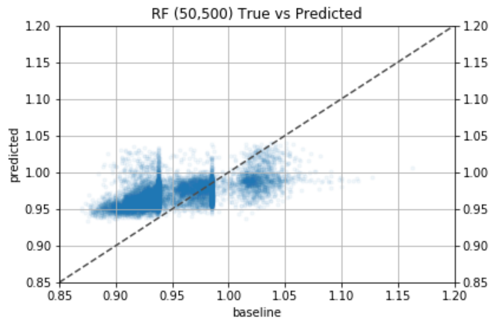


Fig. 10. Predicted values $\hat{y}$ against the actual values y provided by the baseline. Gaussian Process regression and Random Forest at 500 samples actively learnt. A perfect prediction would place all points on the diagonal.

regression algorithms.

Throughout the experiments Gaussian Process regression exhibited the best and also most reliable performance, which is clearly visible in Figure 7. Among the decision tree based regression algorithms Random Forests with maximum depth 50 and 5000 trees performed best.

As a further type of plot, we studied *true-against-predicted* plots (a.k.a predicted-against-actual plots) to gain more qualitative insights into which regression model has learned the function y best. In these plots, we plot for all sampled baseline points X the true value $y(X)$ against the value $\hat{y}(X)$ as predicted by the regression model. Also this investigation suggested that Gaussian Process regression performs best. Refer to Figure 10 for two such plots, where we can see that for Gaussian Process regression the points are closer to the diagonal, i.e., the function was learnt better than for Random Forest.

While Random Forest regression is often the best choice for many machine learning applications, since it is computationally efficient but still yields good results, Gaussian Process regression is clearly preferable for our exploratory testing solution despite the fact that it is computationally much more expensive. The main reason is that when considering the computation time spent on running the simulations to acquire the labels, it is well worth spending more computation time on the actual regression model training.

As a conclusion, we suggest to use Sobol sequence sampling in combination with Gaussian process regressors in order to

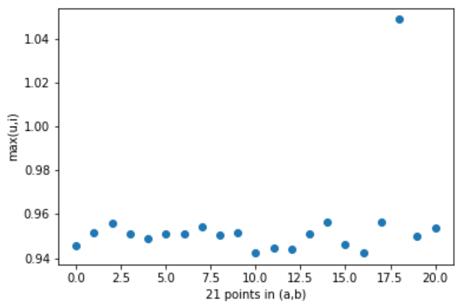


Fig. 11. Small scale sampling along the line between two neighboring baseline points with large difference, $y(a) \approx 0.95$ and $y(b) \approx 1.03$.

perform the exploratory testing for the problem at hand.

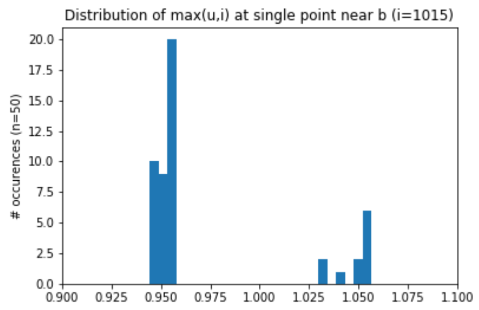
4) Extending the Evaluation to Other Machine Learning Algorithms: As an extension to the presented experiments we investigated several additional regression methods with the goal to push the accuracy further than what Sobol sampling and Gaussian process regression achieve. In these investigations we also relaxed the constraints of limited execution time and allowed up to 5000 baseline samples for learning. We applied Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs) for learning the continuous y as before, and in an additional approach we used SVMs and ANNs as binary classifier and learned y^{τ} directly. In more detail, we used ANNs with the following parameter combinations: 2-4 hidden layers of size 128 or 256; relu or tanh activation functions. Finally, we also applied ordinal regression.

The sobering results of these investigations is that none of the approaches reaches a better accuracy than Gaussian Process regression. While several approaches managed to push the accuracy further for the training set, the test set accuracy stayed close to 90%, i.e., the test data was overfitted. Apparently, our robustness test case turns out to be a relatively hard-to-learn problem.

D. Small Scale Analysis and Stochasticity

The observation that the robustness test case simulation is hard to learn led to follow up investigations with the goal to understand what might cause this function to be hard to learn. To this end, we focused on a point where the difference $|y-\hat{y}|$ between the sampled value and the predicted maximum current/voltage value was particularly large, i.e., $|y-\hat{y}| = 1.03 - 0.95 = 0.08$. When studying the neighboring samples in our baseline, i.e., the points with smallest Euclidean distance, it turns out that all were much closer to 0.95 than 1.03. The question thus is what causes this spike of 1.03. To answer this question we ran a small scale sampling along a line in the parameter space between the observed spike, b, and its closest neighbor a. The result is plotted in Figure 11.

All values stayed between 0.94 and 0.96 except for one spike (point 19). Interestingly, the value at point b was now different from the value in the baseline, namely, $y(b) \approx 0.955$



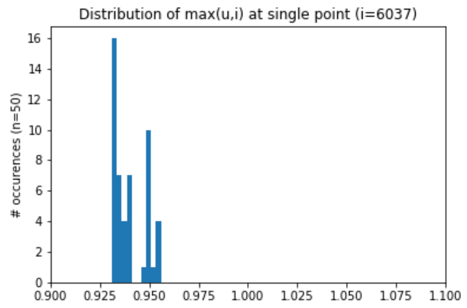


Fig. 12. Distribution of y in a "spiky" area vs. distribution of y at a randomly chosen point.

rather than 1.03. The spike was seen at another point instead, which gave rise to an extended investigation into the stochasticity in the respective area. It turns out that the stochastic component of the simulation is larger than originally assumed and in some areas, like in the one near point b, points exhibit a bi-modal distribution. Figure 12 shows the histogram of the values sampled 50 times at a point near b (point 19) and the values at a single random point in the parameter space for comparison. The stochasticity explains the fact that no regression method yielded a significantly higher accuracy than 90%. With this extended knowledge of the problem, the fact that Gaussian process regression with Sobol sampling achieved a prediction accuracy above 90% already with 250 samples should be interpreted as rather good result.

V. Conclusion and Future Work

This paper presented a solution for integrating automated exploratory testing of industrial systems into a continuous integration workflow. Thereby, the costly simulation of the system is approximated by a regression model, and interesting critical configurations are selected and presented to the engineer with the help of clustering methods. The evaluation results show that while active learning can be beneficial in some cases, Gaussian process regression with Sobol sampling yielded the best results. Consequently, we have chosen Sobol sampling with Gaussian process regression as the default configuration of our testing framework. The framework is now used to run the automated exploratory testing workflow as part of continuous integration.

The analysis in this paper assumes that the input and output parameter space are specified by the end customer with clear boundary conditions. The resulting control system is only checked for these conditions. However, the customers are not always able to describe boundary conditions with the required precision. This is because the control systems can be part of another system and the boundary conditions can slightly be offset due to effect of the other subsystems in the system. This can cause the control system to not work as desired. To this end, future work shall look into estimating the Pareto front of system stability versus parameter values. This will provide insights and estimates on the parameter ranges and their effect on the stability of the resulting control system.

Another interesting question for future work is whether there are transfer learning techniques that could help reusing the regression model learnt in the previous cycle of our workflow, rather than discarding it and learning the function representing the updated test case function from scratch.

References

[1] B. R. Abdessalem, S. Nejati, C. L. Briand, and T. Stifter. Testing advanced driver assistance systems using multi-objective search and neural networks. *ASE*, pages 63–74, 2016.

[2] C. Budnik, M. Gario, G. Markov, and Z. Wang. Guided test case generation through ai enabled output space exploration. In *2018 IEEE/ACM 13th International Workshop on Automation of Software Test (AST)*, pages 53–56, 2018.

[3] S. Burhenne, D. Jacob, and G. P. Henze. Sampling based on sobol'sequences for monte carlo techniques applied to building simulations. In *Proc. Int. Conf. Build. Simulat*, pages 1816–1823, 2011.

[4] T. Danka and P. Horvath. modal: A modular active learning framework for python. *arXiv preprint arXiv:1805.00979*, 2018.

[5] P. Duvall, S. M. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series. Addison-Wesley, 2007.

[6] R. Florea and V. Stray. A qualitative study of the background, skill acquisition, and learning preferences of software testers. In J. Li, L. Jaccheri, T. Dingsøyr, and R. Chitchyan, editors, *EASE '20: Evaluation and Assessment in Software Engineering, Trondheim, Norway, April 15-17, 2020*, pages 299–305. ACM, 2020.

[7] A. Gula, C. Ellis, S. Bhattacharya, and L. Fiondella. Software and system reliability engineering for autonomous systems incorporating machine learning. In *2020 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–6, 2020.

[8] J. Itkonen, M. Mäntylä, and C. Lassenius. The role of the tester's knowledge in exploratory software testing. *IEEE Trans. Software Eng.*, 39(5):707–724, 2013.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] N. Ramli, R. R. Othman, A. Khalib, I. Zahereel, and M. Jusoh. A review on recent t-way combinatorial testing strategy. *MATEC Web of Conferences*, 140:01016, 01 2017.

[11] J. Schreiter, D. Nguyen-Tuong, M. Eberts, B. Bischoff, H. Markert, and M. Toussaint. Safe exploration for active learning with gaussian processes. In *ECML 2015*, 2015.

[12] B. Settles. From theories to queries: Active learning in practice. In Isabelle Guyon, Gavin Cawley, Gideon Dror, Vincent Lemaire, and Alexander Statnikov, editors, *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, volume 16 of *Proceedings of Machine Learning Research*, pages 1–18, Sardinia, Italy, 16 May 2011. PMLR.

[13] M. Thangiah and S. Basri. A preliminary analysis of various testing techniques in agile development - a systematic literature review. pages 600–605, 08 2016.