Test Suite Prioritization for Efficient Regression Testing of Model-based Automotive Software

Andrey Morozov, Kai Ding, Tao Chen, Klaus Janschek
Technische Universität Dresden, Faculty of Electrical and Computer Engineering, Institute of Automation, 01062 Dresden, Germany
Email: {andrey.morozov, kai.ding, tao.chen1, klaus.janschek}@tu-dresden.de

***Abstract*—Up to 80% of the automotive software can be generated from models. MATLAB Simulink is a common tool for creation of complex combinations of block diagrams and state machines, automated generation of executable code, and its deployment on a target ECU. The automotive safety standards require extensive testing of the developed models. Regression testing should be undertaken every time a model is updated to ensure that the modifications do not introduce new faults into the previously validated model. A common, time-consuming way is to rerun an entire test suite after even minor changes. This paper introduces a new method for automatic prioritization of test cases. The method is based on two principles: (i) A test case should stimulate an error in an updated block and (ii) the stimulated error should propagate to the place where it can be detected. The proposed method includes the evaluation of input vectors that are provided to updated blocks by each test case and a Markov-based stochastic error propagation analysis of the model. The application of the method is demonstrated with a Simulink model of a gearbox and a test-suite, automatically generated with the Reactis Tester.**

***Keywords*-Model-based software; Automotive software; Simulink; Software testing; Regression testing; Test suite prioritization; Model-based testing; Error propagation analysis; Control flow; Data flow; Dual-graph error propagation model; Fault activation analysis; Markov chains;**

I. Introduction

Model-based development is an efficient, reliable, and cost-effective paradigm to design and implement complex embedded systems. The software determines more than 90% of the functionality of automotive systems and up to 80% of the automotive software can be automatically generated from models [1], [2]. Software components are no longer handwritten. Advanced toolsets like MATLAB Simulink/Stateflow (MathWorks) and TargetLink (dSPACE) allow an engineer to create complex functional models, generate AUTOSAR compliant production code, and directly deploy it on target ECUs (Electronic Control Units) [3], [4], [5]. Complementary tools like SystemDesk provide the means for creation of sophisticated Composition Diagrams that model integration and interaction of several ECUs into a complete system [6].

The automotive safety standard ISO26262 [7] requires extensive testing of the developed models with numerous test cases including functional tests, integration tests, and structural coverage tests. Testing activities can account a considerable part of the software production costs. The model-based software development allows validation and verification in very early stages of the development process. Software tools like Simulink Test and Verification&Validation (MathWorks), Reactis (Reactive Systems), ECU-TEST (TraceTronic) and TPT (PikeTec) automatically generate test suites for Simulink models trying to achieve maximum coverage and minimum redundancy.

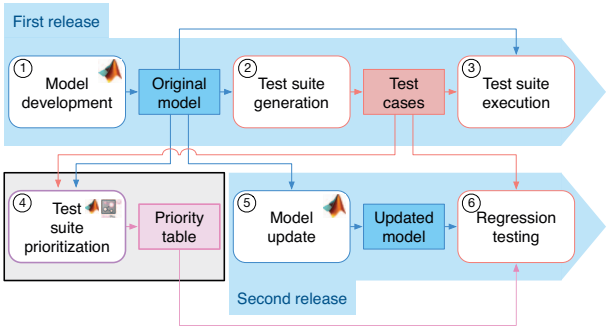

Figure 1. An example of a model-based software development workflow that includes the proposed *Test suite prioritization* method (4). The outcome of the method is the Priority table that helps to speed up the Regression testing (6).

Regression testing is a special type of testing that is undertaken every time a model is updated. The goal of the regression testing is to ensure that the modifications do not introduce new faults into the previously validated model. A common way, the rerunning an entire test suite after even minor changes in the model, requires high repeated effort and significantly increases the testing time.

This paper introduces a new method for automatic prioritization of test cases of an original test suite for efficient regression testing. Figure 1 describes an example of an intended development workflow that includes the application of the proposed method. The workflow consists of two iterations that are highlighted in blue: *First release* and *Second release*. The first iteration of the development process comprises (1) *Model development*, (2) *Test suite generation*

978-1-5386-3687-9/17 $31.00 © 2017 IEEE
DOI 10.1109/SATE.2017.11

that can be either manual or automatic, and (3) *Test suite execution*. The second iteration consists of (5) *Model update* and consecutive (6) *Regression testing*.

The presented technical implementation of our method is based on the following assumptions: (i) the model update is minor, i.e. the structure is unchanged, no new or deleted blocks, only parameters of several blocks are updated and (ii) the original *Test cases* can be reused for the regression testing. Elimination of these assumptions is possible, but requires considerable technical effort. Also, the key goal of this paper, to demonstrate how the fault activation and error propagation analysis supports model-based regression testing, can be achieved with these limitations.

The available time interval between the first and second iterations can be used for automatic (4) *Test suite prioritization*. The proposed method analyzes the *Original model* and the *Test suite* and identifies which test cases should be run first after an update of a particular block. The result of the method application is the *Priority table* that helps to speed up the (6) *Regression testing*.

Automotive companies tend towards continuous integration and deployment of model-based ECU software that can consist of several hundreds of Simulink models. Particularly, we assume a scenario when a developed and tested model is integrated into a new software system and therefore has to be tuned e.g. parameters of a PID controller, a code of an S-Function, or values of a Look-up table should be updated.

Contribution: This paper demonstrates how the combination of fault activation and error propagation analysis methods can support and speed up model-based regression testing of Simulink models with the test suite prioritization. The key contributions of this paper are: (i) a new model-based test suite prioritization method for regression testing of Simulink models that is based on stochastic fault activation and error propagation analysis, (ii) a method for estimation of the fault activation probabilities that examines the input vectors provided to the blocks of the model by each test case, (iii) an adaptation of a comprehensive Markov-based approach to error propagation analysis for the estimation of the likelihood that an error will propagate from an updated faulty block to the place where it can be detected.

The rest of the paper is organized as follows. Section II discusses the relevant available approaches to the regression testing and test case generation software tools. Section III introduces a reference Simulink model. The proposed method is addressed in Section IV. Section V presents the results of the application of the method to the reference model. Conclusions are outlined in Section VI.

II. State of the art

The method, presented in this paper, is oriented, but not limited, to the fast emerging model-based automotive software development domain. A global study that examines the costs and benefits of the model-based development of embedded systems in the automotive industry is presented in [8]. Important characteristics of automotive model-based development processes and the need for the testing in early stages are discussed in [1]. This section gives an overview of the modern regression testing methods and tools and identifies the place of the proposed method according to the classification introduced in [9].

A. Regression testing

Retest-all is the most common and widely used regression testing strategy: The entire test suite is rerun completely. It is the safest, but the most expensive approach. In contrast, a number of more sophisticated techniques are proposed. The three major groups are introduced in [9]:

1) *Test suite minimization*, also called *test suite reduction*, aims to delete redundant and irrelevant test cases in order to reduce the test suite size.
2) *Test case selection*, also called *regression test selection*, identifies a subset of the test cases that are relevant to the update. Contrary to the *test suite minimization*, this technique does not remove test cases permanently.
3) *Test case prioritization* schedules the order of test cases in order to maximize certain properties, such as fault detection or coverage rates.

The *test suite minimization* and *test case selection* are discussed and compared in [10]. A number of *test case selection* techniques for various categories of software are outlined and reviewed in [11] and [12]. An interesting automotive-related signal-trace-based test selection technique for system-level functions is proposed in [13].

B. Test case prioritization methods

The method presented in this paper belongs to the third group, *test case prioritization*. The *test case prioritization* was first mentioned by Wong in [14] and significantly evaluated in a more general context by Harrold and Rothermel in [15], [16]. An extensive survey [17] investigates 120 different test suite prioritization methods following the classification presented in [9]. In addition to [17], a test case prioritization method, based on user's requirements correlation, is presented in [18]. In [19], the authors introduce a value-driven approach to system-level test case prioritization. In [20], an approach, based on the coverage requirements is presented.

According to the classification, introduced in [9], our method falls into the group of *model-based approaches*. Most of the methods in these group are introduced by Korel et al. [21], [22], [23]. The *model dependence-based test prioritization* method [24], [25], [26] analyzes control and data dependencies using the Extended Finite State Machine model of a system under test in order to identify unique interaction patterns between the transitions of an original model and added/deleted transitions of a modified model.

Our literature overview reveals only a few papers about regression testing and test suite prioritization that are focused on Simulink models [27], [28], [29]. An interesting method [29] based on a formal meta-model called Simulink Dependency Graph (SLDG) and a model slicing technique [30] is introduced by Prasad et. al.

Our method comprehensively analyzes and models error propagation processes which are either completely ignored or addressed in a shallow manner (e.g. with SLDGs) by the other methods. An underlying stochastic Dual-graph Error Propagation Model (DEPM) [31] captures control and data flow transitions between Simulink block functions and the probabilities of error propagation through them. For example, our method is sensitive to multi-rate models where a single erroneous output of a block may either result in several erroneous outputs of the model or will be overwritten and the error will not reach the place where it can be detected. Our method also distinguishes between temporary and state variables: If an error reaches a state variable all the further model outputs might be corrupted.

C. Tools for automatic test suite generation

In order to demonstrate the possible integration of our method into an industrial development workflow we have compared several available test suite generators:

Reactis Tester (Reactive Systems) automatically generates test cases that stress a Simulink model and uncovers runtime errors [32]. The generated tests aim to maximize coverage with respect to a number of test coverage metrics and user-defined targets.

TPT (PikeTec) is another software tool for automotive model-based testing [33]. TPT supports structural tests including automatic test case generation for Simulink and TargetLink models.

ECU-TEST (TraceTronic) is a test automation software for embedded automotive systems. This software supports MiL, SiL, and HiL tests of ECUs [34].

Simulink Verification and Validation (MathWorks) is a "native" tool that automates requirements tracing, modeling standards compliance checking, coverage measurements, and test case generation [35].

SimCoTest (University of Luxembourg), is an academic tool that generates small test suites [36]. SimCoTest uses meta-heuristic search to maximize the presence of specific failure patterns in output signals and the diversity of output signal shapes.

Due to the advantage of the flexible test suite generation based on user-specified requirements, we used the *Reactis Tester* for the automatic test suite generation. However, the proposed method can be adapted to any other test suite format that contains the vectors of input and reference output singles or formally-defined assertions.

III. Reference system

This section introduces a Simulink model that will be used throughout the paper as a reference model in order to explain important details of the proposed method.

A. Simulink model

Figure 2 shows a Simulink model of a gearbox. This is a part of a standard Mathworks example of Simulink and Stateflow model of an automatic transmission controller [37]. The model consists of the main system *Transmission* and two subsystems *TorqueConverter* and *TransmissionRatio*. The model has three inputs, engine speed Ne, gearbox position $Gear$, and transmission output speed $Nout$, and two outputs, impeller torque Ti and transmission output torque $Tout$.

Three lookup tables, highlighted in red in Figure 2, are of particular interest: *FactorK*, *TorqueRatio*, and *GearRatio*. In contrast to the other trivial blocks, these lookup tables are likely to be updated during the next release. Therefore, we assume that these three "complex" blocks of interest potentially contain faults.

The *TorqueConverter* and *TransmissionRatio* subsystems operate with two different sample times: One and two time units respectively. This is shown with the red and green colors in the sample time legend in the Figure 2. This difference in the sample times strongly influences error propagation processes. For instance, it can happen that an erroneous output of the *GearRatio* block of the *TransmissionRatio* subsystem will not propagate to the Nin input of the *TorqueConvertor* subsystem. The *Rate Transition* block performs the zero-order hold of the Nin output of the *TransmissionRatio* subsystem and only each second value is used by the blocks of the *TorqueConvertor* subsystem. This can result in the situation where an erroneous value will be rewritten by a correct value on the next simulation step. In contrast, an erroneous output of the *FactorK* or *TorqueRatio* blocks of the *TorqueConvertor* subsystem will be read by the *TransmissionRatio* subsystem two times, resulting in two erroneous values of the $Tout$ output.

B. Test suite generation and testing

A test suite was automatically generated for this model using the *Reactis Tester* software. The exemplary test suite contains 10 test cases and each test case defines a sequence of values for the model inputs: Ne, $Gear$, and $Nout$. In order to keep the values within the assumed operational conditions, the following user-defined targets were specified: $Ne \in (0, 6000)$, $Gear \in [1, 4]$, $Nout \in (0, 6000)$, $Nin/Ne \in (0, 1]$

The *Reactis Tester* allows a user to define a number of assertions that are expected to be true for each test case. An assertion violation during a test case run means that the test case has detected an error. Three assertions were defined for the parts of the model that are highlighted in yellow in

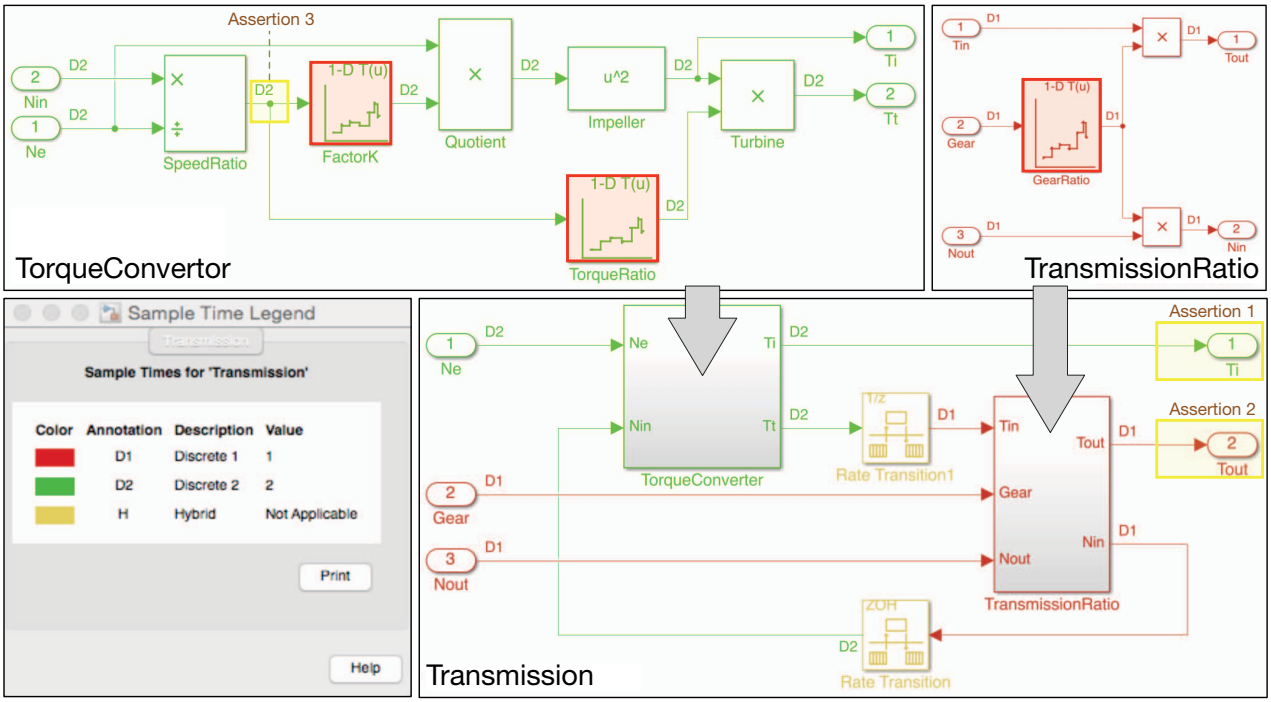

Figure 2. A reference Simulink model of a gearbox. Two subsystems *TorqueConverter* and *TransmissionRatio* operate with different sample times. Three lookup tables *FactorK*, *TorqueRatio*, and *GearRatio*, highlighted in red, are considered to be potentially updated. Three assertions are defined for the parts of the Simulink model that are highlighted in yellow.

Figure 2. Assertions 1 and 2 check that the model outputs Ti and $Tout$ are within their operational intervals. Assertion 3 checks that the speed ratio Nin/Ne is between zero and one.

Assertion 1 $(Ti > 0)$ && $(Ti < 2000)$
Assertion 2 $(Tout > 0)$ && $(Tout < 7000)$
Assertion 3 $(Nin/Ne > 0)$ && $(Nin/Ne \leq 1)$

Following the workflow that is shown in Figure 1, the original Simulink model have been tested with the generated test suite using the defined assertions. After that, we applied our test suite prioritization method. Finally, in order to verify the achieved results, we emulated an update of the original model with minor modifications of the breakpoint parameters of all three lookup tables and performed fault injection experiments with the updated model.

IV. Test Suite Prioritization Method

The proposed prioritization method is based on two key principles: (i) A test case should stimulate an error in an updated block (fault activation) and (ii) the stimulated error should propagate to the place where it can be detected (error propagation). Figure 3 shows the corresponding decomposition of the method into two parts: (1) *Fault activation analysis* and (2) *Error propagation analysis*. Technically, the prototype of the method is implemented as a collection of MATLAB scripts. The first part exploits the Simulink API in order to analyze the behavior of the *Original model* using the input vectors of the *Test cases* and evaluate the *Fault activation probabilities* for each block of interest for each test case. The second part implements an interface with our Markov-based method for error propagation analysis, adapted to the test suite prioritization task, that allows the computation of the mean number of errors that will propagate from outputs of a particular block of interest to the assertions.

A. Fault activation analysis

Our MATLAB script automatically instruments the Simulink model in such way that the input values of the potentially faulty blocks are stored and analyzed using the three following quantitative metrics.

Number of unique input values (M_1): We assume that each unique input has a certain probability of stimulating an error in an updated block. Therefore, the higher is the number of unique inputs that are provided by a particular test case, the higher is the probability that this test case will stimulate an error:

$$M_1 = N_i / N_{all}$$

$M_1 \in [0, 1]$, N_i is the number of unique input values of the

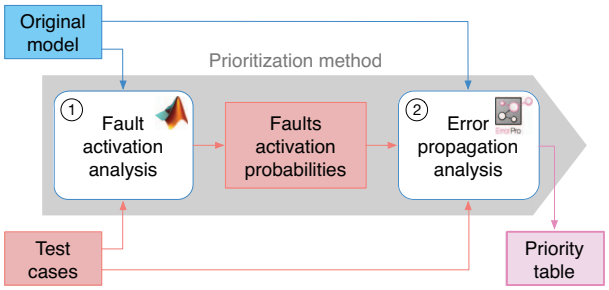

Figure 3. The proposed prioritization method consists of two parts: (1) Fault activation analysis and (2) Error propagation analysis.

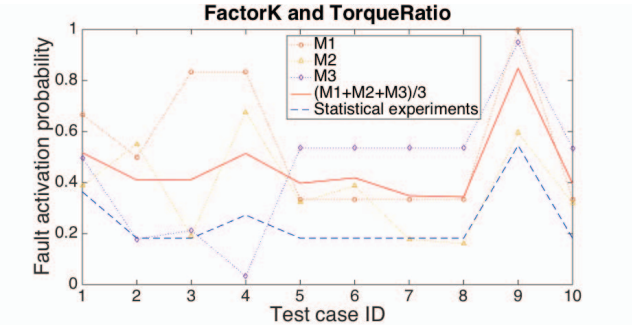

Figure 4. The estimated probabilities of fault activation and the results of the statistical fault injection experiments for the FactorK and TorqueRatio blocks for each test case.

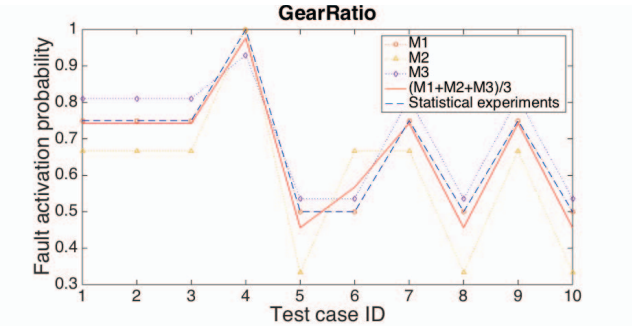

Figure 5. The estimated probabilities of fault activation and the results of the statistical fault injection experiments for the GearRatio block for each test case.

potentially faulty block provided by the i^{th} test case and N_{all} is the number of unique inputs provided by all test cases.

Length of the coverage interval (M_2): We also assume that the bigger is the interval, covered by the input values of a test case, the higher is the fault activation probability:

$$M_2 = (v_i^{max} - v_i^{min})/(v_{all}^{max} - v_{all}^{min})$$

$M_2 \in [0,1]$ and v_i^{max}, v_i^{min}, v_{all}^{max}, and v_{all}^{min} are the maximum and minimum values provided by the i^{th} test case and all test cases respectively.

Distribution of the values (M_3): The last metric shows how the values are distributed within the coverage interval. We assume that the closer is the distribution of the values to the uniform distribution, the higher is the fault activation probability:

$$M_3 = ks\_test(V_i, V_{uf})$$

$M_3 \in [0,1]$ shows the result of the comparison of the set of input values V_i provided by the i^{th} test case with a uniformly distributed set of values V_{uf} for the same coverage interval using the *Kolmogorov-Smirnov Test* method [38]. The higher is the value of M_3, the closer is the distribution to the uniform distribution.

M_1, M_2, and M_3 are combined together in order to estimate the *Fault activation probabilities* for the second part of the proposed prioritization method (see Figure 3) using the following "first guess" combination function:

$$P_{FA} = (M_1 + M_2 + M_3)/3$$

More precise correlation analysis may help to estimate a better form and parameters of the combination function. However, even this very basic function gives acceptable results for our reference model.

The estimated *Fault activation probabilities* as well as the values of the metrics M_1, M_2, and M_3 are presented in Figures 4 and 5. Figure 4 shows the results of the analysis of input vectors provided by the 10 generated test cases for the *FactorK* and *TorqueRatio* blocks, which receive the same inputs. Figure 5 shows the results for the *GearRatio* block.

A number of statistical fault injection experiments have been performed in order to verify the first part of the method. The *FactorK* and *TorqueRatio* blocks are 1-D lookup tables with 11 breakpoints and the *GearRatio* block is also a 1-D lookup table with 4 breakpoints. We emulate faulty parameterization of these lookup tables by replacing a single breakpoint parameter with an erroneous value (zero) and checking, which test cases will stimulate this fault. Overall, 11 different faulty versions of *FactorK* and *TorqueRatio* and 4 faulty versions of *GearRatio* have been generated. The red solid curves in Figures 4 and 5 show the analytically estimated fault activation probabilities using the discussed combination of the metrics and the blue dashed curves show the results of the statistical experiments.

The high correlation of the estimated and statistical results proves our assumption on the validity of the selected metrics and the combination function at least for lookup tables.

B. Error propagation analysis

The analysis of error propagation from the potentially faulty blocks to the assertions, based on the estimated fault activation probabilities, is the second part of the proposed test case prioritization method. We implemented a new interface and adapted our method for stochastic error propagation

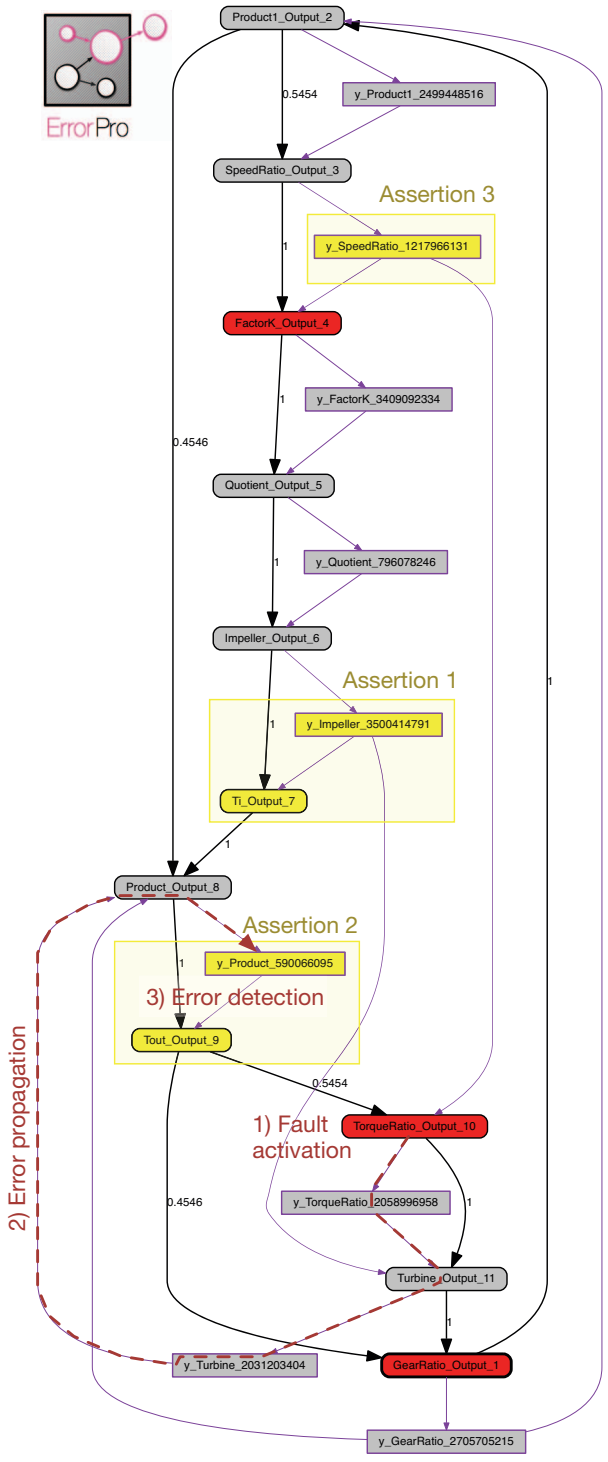

Figure 6. The dual-graph error propagation model that was automatically generated from the reference Simulink model. The rounded gray rectangles represent Simulink block functions. Black arrows represent probabilistic control flow transitions between the block functions. Purple rectangles and arrows represent data storages and data flow arcs respectively. The block functions of the potentially faulty blocks are highlighted in red and the data storages associated with the assertions are highlighted in yellow.

analysis of Simulink models, introduced in [39], to this particular test case prioritization task.

The error propagation analysis part consists of two main steps: (i) automatic transformation of a Simulink model into the formal *Dual-graph Error Propagation Model* (DEPM) and (ii) Markov-based numerical analysis of the generated DEPM.

The DEPM is a mathematical abstraction of system design aspects that influence error propagation processes: control flow, data flow, and component-level reliability properties. The formal set-based notation of the DEPM is as follows:

$$DEPM := \langle E, D, A_{CF}, A_{DF}, C \rangle$$

- E is a non-empty set of executable elements;
- D is a set of data storages;
- A_{CF} is a set of directed control flow arcs;
- A_{DF} is a set of directed data flow arcs;
- C is a set of conditions of the elements.

Figure 6 shows a DEPM that was automatically generated from the reference Simulink model. The rounded gray rectangles represent the executable system elements E. In particular case, Simulink block functions play the role of the DEPM elements. The elements are interconnected by the control flow arcs A_{CF} (black lines). Different sample times of the subsystems of the Simulink model result in control flow branching. The elements are also connected with the data storages D (purple rectangles) by the data flow arcs A_{DF} (purple lines).

Besides the structural system description, the DEPM also contains stochastic properties: probabilities of control flow transitions, probabilities of errors propagation through system elements, and probabilities of fault activation in the elements during their execution. The fault activation and error propagation probabilities are defined using the probabilistic conditions C that specify the conditional probabilities of different erroneous and error-free combinations of outputs for different erroneous and error-free combinations of inputs for each element from E.

The functions of the potentially faulty blocks have non-zero fault activation probabilities. They are highlighted in red in Figure 6. The data storages that are associated with the assertions are highlighted in yellow. The data flow arcs define possible paths of error propagation, e.g. from the *TorqueRatio* block function to the *Assertion 2*, see the red dashed line in Figure 6. The stochastic properties allow the numerical computation of the error propagation probabilities through these paths.

The transformation of a Simulink model into a DEPM model consists of 3 steps [39]: (i) control flow analysis, (ii) data flow analysis, and (iii) block-level analysis. The Simulink API allows running user-implemented call-backs before and after the execution of each block function. We use this mechanism in order to instrument the model

and gather the information about the execution order of the block functions. This information is transformed into the set of control flow arcs and control flow transition probabilities A_{CF}. The data flow analysis is based on the static parsing of a Simulink model. Our algorithm analyzes signal flow between the output and input ports of the blocks, taking into account their state variables. The block-level analysis involves individual simulations for each non-virtual Simulink block in order to estimate the probabilities of error propagation between the input, state, and output variables of the blocks.

The probabilities of a fault activation during a single execution of a block function of a potentially faulty block are computed using the formula:$p_{FA} = 1 - \sqrt[n]{(1 - P_{FA})}$. Where P_{FA} is the probability of a fault activation during the execution of a test case that we have evaluated in the previous step, and n is the number of executions of this block function during the run of the particular test case.

ErrorPro™[40] is our analytical software tool that is based on the described DEPM. This tool automatically creates a set of discrete time Markov chain (DTMC) models using the DEPM model and computes the mean number of errors that will reach the selected data storages, associated with the assertions. Each state of the generated DTMC models contains the information about the element that will be executed in the next simulation step and current erroneous or error-free states of all data storages [41]. The computation of the DTMC models is implemented via an interface with the PRISM model checker [42].

We have performed the error propagation analysis separately for each test case using the corresponding fault activation probabilities, evaluated during the previous step. The numerical allows the test suite prioritization according to the following rule: The higher is the estimated number of errors in the data storages, associated with the assertions, the higher is the priority of the particular test case.

V. Method application results

Our prototypical software implementation has been run on a regular PC and the execution took several minutes for the reference Simulink model (see Figure 2). The results of the application of the introduced method are presented in Tables I, II, and III. The test cases are sorted according to the estimated cumulative number of errors stimulated by the test cases and propagated to the assertions. For example, the Table I shows that in the case of an update in the *FactorK* block the regression testing has to be started with the test cases TC9, TC1, and TC4. Tables II and III give the similar information for the *TorqueRatio* and *GearRatio* blocks respectively.

The error propagation part of the method helps to identify which assertions are reasonable for efficient regression testing of each potentially faulty block. For example, Table I shows that the errors from the *FactorK* block can propagate only to the Assertions 1 and 2 and the number of errors that will reach the Assertion 2 is higher. Table II shows that the errors from the *TorqueRatio* block can propagate only to the Assertion 2. Table III shows that the errors from the *GearRatio* block can propagate only to the Assertions 2 and 3 and the number of errors that will reach the Assertion 2 is higher. This allows concluding that Assertion 2 is the most critical for the regression testing of all three potentially faulty blocks.

The most right columns of Tables I, II, and III show the results of the error injection experiments that have been carried out in order to verify the introduced method. The *Percentage of detected errors* show how many injected errors have been detected by each test case during the statistical experiments. The same fault injection method as for the verification of the fault activation part has been used. The results reveal the strong correlation between the estimated number of errors and the percentage of the detected errors during the experiments.

Average Percentage of Fault Detection (APFD) [43] is a popular metric for the evaluation of the efficiency of test suite prioritization methods. Figures 7, 8, and 9 show the APFD plots for the original and prioritized test suites. The horizontal axes of the plots show the number of executed test cases. The labels near the curves show the test case IDs. The testing process goes from left to right. The vertical axes show the cumulative number of detected faults. The black solid curves are plotted for original test suits and the blue dashed curves for the prioritized ones. For instance, Figure 7 show that 6 out of 11 faults of the *FactorK* block will be detected after the execution of the three first test cases according to the original order (TC1, TC2, TC3) and 8 faults will be detected after the execution of the three first test cases of the prioritized test sequence (TC9, TC1, TC4).

An APFD value is computed as the ratio of the area under the curve to the entire area of the plot. The larger is the area under the curve, the better is the prioritization. The APFDs for the *FactorK* and *TorqueRatio* blocks are 58.64% and for the *GearRatio* block is 87.50%. The APFDs for the prioritized test sequences for the *FactorK* and *TorqueRatio* blocks are 76.82% and for the *GearRatio* block is 95%. The gain of the APFD after the prioritization equals to 18.18% for the *FactorK* and *TorqueRatio* blocks and 7.5% for the *GearRatio* block.

VI. Conclusion

A new model-based method for test case prioritization has been introduced in this paper. The method can be applied automatically after the development of the original Simulink model in order to prepare priority tables that will help to speed up future regression testing. The method consists of two parts: fault activation analysis and error propagation analysis. The first part is based on the evaluation of input vectors that are provided to the blocks of the model by each

Table I
THE PRIORITY TABLE FOR THE FACTORK BLOCK.

Test case ID	Estimation: Number of errors propagated to the assertions (A1, A2, A3): (A1)	(A2)	(A3)	Estimation: Test case priority: (Sum)	Experiments: Percentage of detected errors:
TC9	1.596	2.216	0.0	3.812	55%
TC1	0.676	0.939	0.0	1.615	36%
TC4	0.671	0.932	0.0	1.603	27%
TC6	0.512	0.711	0.0	1.223	18%
TC3	0.502	0.697	0.0	1.199	18%
TC2	0.499	0.691	0.0	1.190	18%
TC5	0.479	0.666	0.0	1.145	18%
TC10	0.477	0.663	0.0	1.140	18%
TC7	0.408	0.567	0.0	0.975	18%
TC8	0.401	0.557	0.0	0.958	18%

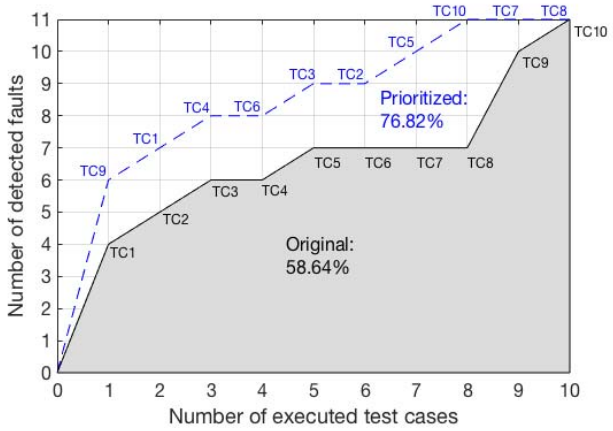

Figure 7. The APFD metrics for the original and prioritized test sequences for the faults of the FactorK block.

Table II
THE PRIORITY TABLE FOR THE TORQUERATIO BLOCK.

Test case ID	Estimation: Number of errors propagated to the assertions (A1, A2, A3): (A1)	(A2)	(A3)	Estimation: Test case priority: (Sum)	Experiments: Percentage of detected errors:
TC9	0.0	2.465	0.0	2.465	45%
TC4	0.0	1.028	0.0	1.028	27%
TC1	0.0	1.006	0.0	1.006	36%
TC3	0.0	0.806	0.0	0.806	18%
TC6	0.0	0.782	0.0	0.782	18%
TC2	0.0	0.769	0.0	0.769	18%
TC5	0.0	0.733	0.0	0.733	18%
TC10	0.0	0.730	0.0	0.730	18%
TC7	0.0	0.625	0.0	0.625	18%
TC8	0.0	0.613	0.0	0.613	18%

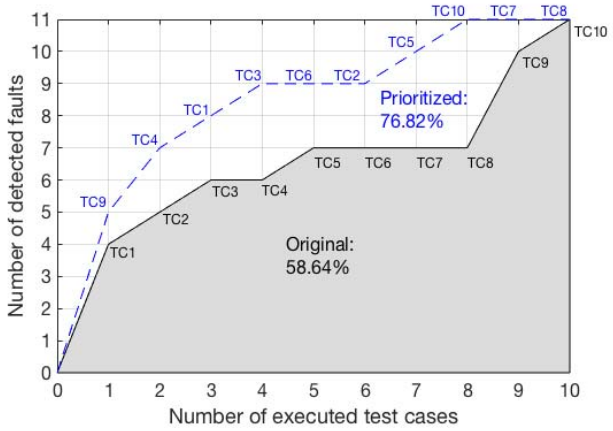

Figure 8. The APFD metrics for the original and prioritized test sequences for the faults of the TorqueRatio block.

Table III
THE PRIORITY TABLE FOR THE GEARRATIO BLOCK.

Test case ID	Estimation: Number of errors propagated to the assertions (A1, A2, A3): (A1)	(A2)	(A3)	Estimation: Test case priority: (Sum)	Experiments: Percentage of detected errors:
TC4	0.0	3.150	1.773	4.923	100%
TC1	0.0	1.264	0.711	1.975	75%
TC2	0.0	1.264	0.711	1.975	75%
TC3	0.0	1.264	0.711	1.975	75%
TC7	0.0	1.264	0.711	1.975	75%
TC9	0.0	1.264	0.711	1.975	75%
TC6	0.0	0.799	0.450	1.249	50%
TC5	0.0	0.587	0.330	0.917	50%
TC8	0.0	0.587	0.330	0.917	50%
TC10	0.0	0.587	0.330	0.917	50%

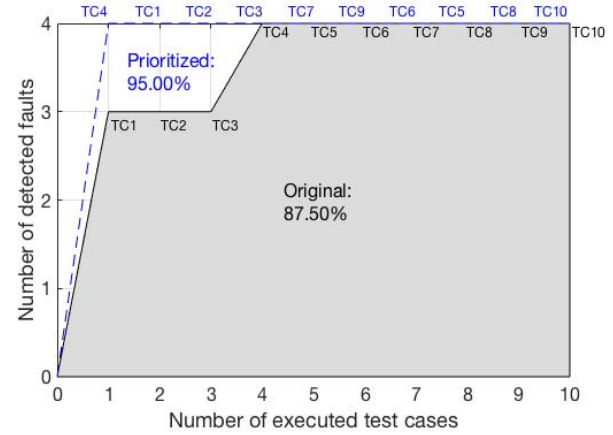

Figure 9. The APFD metrics for the original and prioritized test sequences for the faults of the GearRatio block.

test case. This evaluation helps to estimate the probability that a test case will stimulate an error in a particular block. The second part is based on the method for stochastic error propagation analysis that was adapted for the test suite prioritization task. This method predicts how many stimulated errors will propagate to assertions, associated with the test cases, and become detectable. A prototype of the method has been implemented and demonstrated using a Simulink model of an automotive transmission.

The introduced method requires several further improvements. The fault activation part should be elaborated in order to cover not only lookup tables but also other types of Simulink blocks. We should consider that several high priority test cases may uncover the same faults. For instance, an additional metric that measures the similarity of the input vectors of different test cases can be used. The method should be applied for a case study where the input vectors of different test cases significantly influence the control flow of the block functions e.g. the combination Simulink and Stateflow models. in this case, the impact of the error propagation analysis part on the test suite prioritization will be much higher. The method requires performance evaluation with a large-scale case study.

ACKNOWLEDGMENT

This work is supported by the German Research Foundation (DFG) project JA 1559/5-1.

REFERENCES

[1] E. Bringmann and A. Krämer, "Model-based testing of automotive systems," in *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 2008, pp. 485–493.

[2] S. Wagner, F. Deissenboeck, S. Teuchert, and J.-F. Girard, *Model-Driven Software Development: Integrating Quality Assurance*. Idea Group, 2008, ch. Assuring Maintainability in Model-Driven Development of Embedded Systems. [Online]. Available: http://www.irma-international.org/viewtitle/26836/

[3] MathWorks, "Matlab & simulink: Simulink users guide r2017a," 2017.

[4] ——, "Matlab & simulink: Stateflow users guide r2017a," 2017.

[5] dSPACE, "Targetlink, retireved october 2016," https://www.dspace.com/en/pub/home/products/sw/pcgs/targetli.cfm, 2016.

[6] ——, "Systemdesk, retireved october 2016," https://www.dspace.com/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm, 2016.

[7] ISO, "ISO 26262, Road vehicles – Functional safety," 2011.

[8] M. Broy, S. Kirstan, H. Krcmar, and B. Schätz, "What is the benefit of a model-based design of embedded software systems in the car industry?" *Software Design and Development: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*, p. 310, 2013.

[9] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[10] A. Shi, T. Yung, A. Gyori, and D. Marinov, "Comparing and combining test-suite reduction and regression test selection," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 237–247.

[11] S. Biswas, R. Mall, M. Satpathy, and S. Sukumaran, "Regression test selection techniques: A survey," *Informatica*, vol. 35, no. 3, 2011.

[12] G. Rothermel and M. J. Harrold, "Empirical studies of a safe regression test selection technique," *IEEE Transactions on Software Engineering*, vol. 24, no. 6, pp. 401–419, 1998.

[13] S. Vöst and S. Wagner, "Trace-based test selection to support continuous integration in the automotive industry," in *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*. ACM, 2016, pp. 34–40.

[14] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," in *Proceedings of the 17th International Conference on Software Engineering*, ser. ICSE '95. New York, NY, USA: ACM, 1995, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/225014.225018

[15] M. J. Harrold, "Testing evolving software," *J. Syst. Softw.*, vol. 47, no. 2-3, pp. 173–181, Jul. 1999. [Online]. Available: http://dx.doi.org/10.1016/S0164-1212(99)00037-0

[16] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*. IEEE, 1999, pp. 179–188.

[17] C. Catal and D. Mishra, "Test case prioritization: a systematic mapping study," *Software Quality Journal*, vol. 21, no. 3, pp. 445–478, 2013.

[18] T. Ma, H. Zeng, and X. Wang, "Test case prioritization based on requirement correlations," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on*. IEEE, 2016, pp. 419–424.

[19] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *2005 International Symposium on Empirical Software Engineering, 2005.* IEEE, 2005, pp. 10–pp.

[20] D. Jeffrey and N. Gupta, "Test case prioritization using relevant slices," in *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, vol. 1. IEEE, 2006, pp. 411–420.

[21] B. Korel, G. Koutsogiannakis, and L. H. Tahat, "Model-based test prioritization heuristic methods and their evaluation," in *Proceedings of the 3rd international workshop on Advances in model-based testing*. ACM, 2007, pp. 34–43.

[22] ——, "Application of system models in regression test suite prioritization," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. IEEE, 2008, pp. 247–256.

[23] B. Korel and G. Koutsogiannakis, "Experimental comparison of code-based and model-based test prioritization," in *Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on*. IEEE, 2009, pp. 77–84.

[24] B. Korel, L. H. Tahat, and B. Vaysburg, "Model based regression test reduction using dependence analysis," in *Software Maintenance, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 214–223.

[25] B. Korel, L. H. Tahat, and M. Harman, "Test prioritization using system models," in *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. IEEE, 2005, pp. 559–568.

[26] L. Tahat, B. Korel, G. Koutsogiannakis, and N. Almasri, "State-based models in regression test suite prioritization," *Software Quality Journal*, pp. 1–40, 2016.

[27] B. Minj, "Generation and prioritization of test casesusing simulink/stateflow models," Ph.D. dissertation, Department of Computer Science and Engineering National Institute of Technology Rourkela Rourkela - 769008, India, 2014.

[28] R. Sharma, "Finding dependency, test sequences and test cases for simulink/stateflow models," Ph.D. dissertation, Department of Computer Science and Engineering National Institute of Technology Rourkela Rourkela 769008, India, 2015.

[29] S. Prasad, S. Nayak, V. Vijay, and R. Mall, "Sldg: a metamodel for simulink/stateflow models and its applications," *Innovations in Systems and Software Engineering*, vol. 12, no. 4, pp. 263–278, 2016.

[30] R. Reicherdt and S. Glesner, "Slicing matlab simulink models," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 551–561.

[31] A. Morozov and K. Janschek, "Probabilistic error propagation model for mechatronic systems," *Mechatronics*, vol. 24, no. 8, pp. 1189 – 1202, 2014.

[32] ReactiveSystems, "Testing and validation of simulink models with reactis," http://www.reactive-systems.com/papers/bcsf.pdf, 2013.

[33] PikeTec, "Tpt - test and verification of embedded control software," https://www.piketec.com/en/, accessed: 2017-04-27.

[34] TraceTronic, "Ecu-test v6.5 - systematic and automatic software testing of ecus," https://www.tracetronic.com/cms/data/docs/pdf/Data_sheet_ECU-TEST.pdf, accessed: 2017-04-27.

[35] MathWorks, "Simulink verification and validation users guide r2017a," 2017.

[36] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Simcotest: a test suite generation tool for simulink/stateflow controllers," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 585–588.

[37] MathWorks, "Modeling an automatic transmission controller," https://www.mathworks.com/examples/simulink/mw/simulink_product-sldemo_autotrans-modeling-an-automatic-transmission-controller, 2016.

[38] F. J. Massey, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951. [Online]. Available: http://www.jstor.org/stable/2280095

[39] A. Morozov, K. Janschek, T. Krüger, and A. Schiele, "Stochastic error propagation analysis of model-driven space robotic software implemented in simulink," in *Third Workshop on Model-Driven Robot Software Engineering, Leipzig, Germany*, 2016.

[40] A. Morozov, R. Tuk, and K. Janschek, "ErrorPro: Software tool for stochastic error propagation analysis," in *1st International Workshop on Resiliency in Embedded Electronic Systems, Amsterdam, The Netherlands*, 2015, pp. 59–60.

[41] A. Morozov and K. Janschek, "Flight control software failure mitigation: Design optimization for software-implemented fault detectors," *IFAC-PapersOnLine*, vol. 49, no. 17, pp. 248–253, 2016.

[42] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[43] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on software engineering*, vol. 27, no. 10, pp. 929–948, 2001.