

Optimizing Regression Testing with Functional Flow Block Reliability Diagram

Vaishali Chourey¹ ^a and Meena Sharma²

¹Department of IT, Medi-Caps University, Indore, India

²Department of Computer Engineering, IET DAVV, Indore, India

Keywords: Regression Testing, UML, Test Prioritization, Reliability Testing.

Abstract: Model based testing techniques are widely used to generate tests from models and their specifications. Regression Testing is critical in evolving software systems. It implies that Software undergoes continuous integration and subsequent testing of modules and components to develop a final release of the software. Recent research on test automation have revealed challenges in testing such as enormous test cases generated that are either redundant or irrelevant tests. This prevents the critical parts of code from testing and leave faults uncovered. Manual testing also slows the process and makes exhaustive testing for quality difficult. This paper proposes an approach to generate regression tests from model synthesized from UML interaction diagrams and also prioritize tests. The algorithm generate test suites from the proposed intermediate model. The reliability analysis is performed on blocks and the results govern the prioritization of tests.


1 INTRODUCTION

Model-based testing (Aichernig et al., 2018) has proved to be an advantageous approach to design tests for evolving and complex software systems. Modern software systems are developed for adaptable services by integration of large number of components. Such software systems actually require systematic, scalable and automated testing support. Another characteristic of such systems is that they need frequent updates to maintain their evolution. In such cases, regression testing becomes a crucial part of the development and maintenance phases. Regression testing verifies the code modified during the update or an upgrade. Regression testing ensures that the update in the system does not alter or deviate the inherent behavior of the system. The test automation also poses serious problems with enormous amounts of test cases generated that are redundant. Software are composed of interacting components which are amenable to testing. Such black box components are interface dependent and their evaluation in combination with other components is difficult. Implementing under the Agile teams, the fast delivery imposes time constraints and the exhaustive testing is not possible. Studies for obtaining maximum coverage and fulfillment of tests

under such constraints are gaining attention in many researches.

Software undergoes modifications with changing user requirements or improvements in the functionalities, handling errors and correcting defects (Dick et al., 2017). This is where regression testing plays an important role for maintaining the quality of the software in the event of changed specifications. The regression testing also concerns for quality in terms of security, reliability, dynamic accessibility and inter-component dependencies (Yoo and Harman, 2012). It ensures changes may not affect the system for these quality issues along with the consistent functional specifications. Automated testing has eased the task of testing but with a price of enormous tests generated as result (Gupta et al., 2018). Prioritizing tests, maximizing coverage with minimum test cases is the aim of regression testing today. This reduces the overall cost and time for testing also. When testing goes along with development, the test case prioritization takes place simultaneous to development of the module and benefits the managers and the testers.

The process of regression testing is an unavoidable activity along with integration during development of system functionalities (Luo, 2001). The test case generation aligns with the development as the cost of test generation at later stage incurs additional cost and efforts. A complete test suite for code, spe-

^a  <https://orcid.org/0000-0002-2171-9247>

cific to the development environment is required before the testing. The effectiveness of tests generated lies in the strategies to minimize the test cases, selecting relevant tests and prioritize tests (Kandil et al., 2017). There are challenges in the regression testing around recursively growing lengthy tests. Manual intervention to stop such tests leads to undefined sections in system with un-approached tests. Tests aborted may leave the failures concealed. Tests are drawn at unit, integration or system levels depending on the scope and extent of testing. Unit testing is mature with tools available for the purpose, specific to the language and application. The difficult part is the system testing which depends on the artefacts, specifications, and other information related to the software system.

UML provides the necessary information about the system, its structure, its interactions and specifications to aid the development and testing (Arora and Bhatia, 2018; Mingsong et al., 2006; Sarma and Mall, 2007b). The methodology proposed in the paper is a strong justification to adopt model based testing as a scope to handle the challenges of testing. With Agile and other development strategies, where design are less relevant, a model is proposed to use model information and facilitate regression testing (Teixeira and e Silva, 2018; Wu et al., 2018). The sequence of interactions is important to understand the behaviour of the system. Testable model, called as Functional Flow Block Reliability Diagram (FFBRD) is hereby proposed to generate the tests from the model and its annotation to estimate the reliability quality attribute. The approach in this paper addresses the issues of test generation, estimation of quality and prioritization of tests.

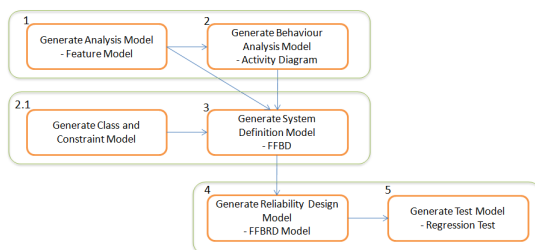


Figure 1: Flow Diagram for FFBRD and Test Generation.

The figure 1 explains the steps in generating the desired intermediate model. The requirements are modelled as structural and behavioral diagrams which acts as input to the FFBRD. The levels of abstractions in defining the components of the system are adapted from feature model and the dynamics is modelled from the activity flow (Chourey and Sharma, 2016). The annotation is formulated as constraints of the classes. Altogether the structural aspect of FFBD

is attained. The expected quality attribute is then annotated to visualize quality in the design. The features, blocks and constraints together formally defines the FFBRD. Simulating the execution of the transactions are studied as operational behavior of the system, the test suites are generated (Kim et al., 2017). The test cases emerging from the transactions are further categorized as critical and non critical. This also enables prioritization of components and flows. The concept of FFBRD enables the use of existing semantics and models from UML and SysML. This best suits the process of test optimization (Felderer and Herrmann, 2018).

The organization of this paper is as follows. The next section is an overview of related work in the area of model based testing, specifically in regression testing. Brief definition and structure of FFBRD is proposed in section 3. Section 3 also presents illustration of approach in system representation and test generation. Section 4 describes proposed algorithm for test generation and prioritization. Last section includes conclusion and scope of the work.

2 RELATED WORK

This section reviews the research developed around the scope of the proposed work. This paper focuses on two aspects - Functional Flow Block Diagrams and Reliability Assessment. FFBD (McInnes et al., 2011) are the formal graphical representations in systems engineering for behaviour of complex hierarchical systems. The software architecture with EFFBD are formally defined in SysML standards (Alanen and Porres, 2005). Blocks defined in SysML has a close resemblance to the components and their flows. The generic definitions and syntax of SysML enables the construction of metamodel for FFBD. The application of FFBD to visualize reliability in design extends the notation to a new vocabulary of FFBRD. This is a combination of flow (dynamics) of UML specifications and analysis of reliability in terms of RBDs. The UML based semantics for software reliability through design is validated with reliability assessment strategies (Gokhale and Trivedi, 2002). The target software model that was a functional specification extends to quality specific attributes. The software model generates test artifacts for regression testing. FFBDs in this semantic make it adaptable to tools and analysis applicable to UML designs. This prevents unambiguous interpretations and fits FFBDs for software design and test modeling.

Reliability assessment models have been proposed to analyze and quantify reliability which is an impor-

tant aspect of software quality (Lyu et al., 1996). An array of methods exists, applicable during different phases of software development life cycle (Goševa-Popstojanova and Trivedi, 2001). These black box techniques depend on failure data for analyses (Pham and Pham, 2017). The types of software developed nowadays need analysis based on modules, their interactions and dynamically making choice of substituting modules for quality improvement. The key identifiers in the process of reliability modeling are:

- Specifying the structure of the system composed of reusable modules/components along with their quality and performance attributes;
- Analyzing reliability with choice of the module to increase overall reliability;
- Analyzing interfaces and interactions across the modules;
- Prioritizing critical modules.

The most interesting part is to bring the FFBD and the reliability testing in the regression test suits. So far, UML models have been an important study for testing functionality (Yildirim et al., 2017) through behavioural diagrams like activity diagram, state charts and sequence diagrams (Teixeira and e Silva, 2018; Sarma and Mall, 2007a). Converting into graphs, scenarios or interpretations from models have aided test generation from models (Jena et al., 2014; Nelson, 2009). The effectiveness is to map it to test for quality attributes in the same manner as functionality is the application of the concept. The simulation of flows in the system makes it amenable to be analysed for its execution time behaviour. This facilitates understanding the performance of the system during the design stage. Such estimations also support optimization for quality-wise delivery of the functionality (Nelson, 1983; Zheng et al., 2017). For the scope of paper, reliability is considered for evaluation of quality. The tool support of ReliaSoftBlockSim enables the analysis of components of the system for reliability. The conversion of behavioural models into blocks and then modelling their transitions for serial or parallel communications derives the system reliability (Chourey and Sharma, 2015). The next section illustrates the concept of FFBRD and the contribution of the paper.

3 PROPOSED METHODOLOGY

The basic structural component of the FFBRD is the Block. Blocks are modular units of system description. They are a collection of structural and be-

havioural features to define a system. A Block is defined with the context of its functionality and quality requirements. The Use cases provide the functional requirements whereas the provision of defining views and viewpoints in UML4SysML provides a generic method to define the quality perspective. Although the concept stems from SysML, the semantics of the diagram is inherited from the class structure and flows through activity diagram.

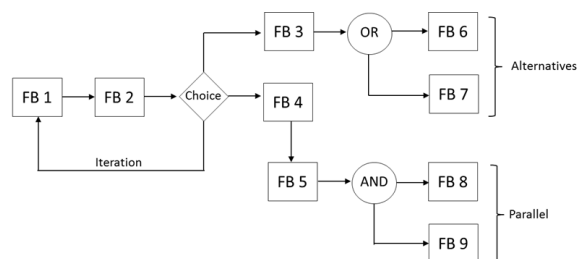


Figure 2: Notation for FFBRD.

FFBRDs can be arranged in a hierarchy to define the levels of abstraction of system understanding as in figure 3. This is a feature diagram that has levels and each level initiates the next level of its definition in case of complex activities or dependent abstractions.

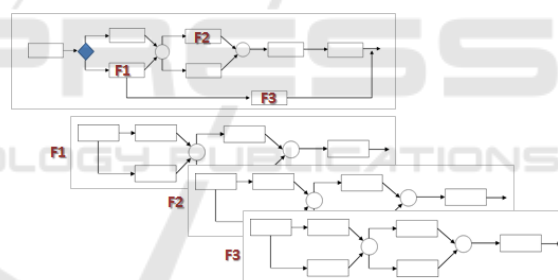


Figure 3: Levels of Abstractions in FFBRD.

The FFBRD contains blocks (FB_i) with communications amongst the blocks and properties (pi) of the block to describe the structure of the system. For a special case the reliability values of the block is annotated in the properties as R_i. The stereotype is Enhanced Functional Flow Block Diagram (EFFBD) with Base Class::UML4SysML::Activity and constraints are applicable in parameters, actions, activity and execution. The constructs of FFBRD is given in the notation of FFBD and is modelled as sequence of blocks, iterations, choice, alternatives and parallel blocks. The flow and order of execution of the blocks as in figure 2 characterizes the behaviour specification in the model.

An example shows a checkout scenario, its equivalent use case and FFBRD to visualize the reliability of the subsystem. The scenario consists of checkout after the purchase. The registered customer up-

Table 1: Formalization of FFBRD in UML and SysML Semantics.

Block Feature	Metamodel Reference from SysML, UML4SysML
Block Definition	SysML::Blocks
Property	UML4SysML::Property
Input	UML4SysML::AcceptEventAction
Output	UML4SysML::SendSignalAction
Control Values	ControlNode and ObjectNode from UML4SysML::ActivityNode
ControlFlow	UML4SysML::ControlFlow
Probability	SysML::Activities::Probability
Reliability	UML4SysML::Reliability extends UML4SysML::ParameterSet
ConstraintBlock	SysML::ConstraintBlocks

dates his cart and makes payment for the calculated amount. The payment interface accepts payment options to proceed. The use case diagram enumerates the functionalities in the scenario. The flow of control and data is formulated as FFBD as shown in the figure 4. An implementation of the model in SysML is supported in Papyrus tool in Eclipse environment (Gérard,).

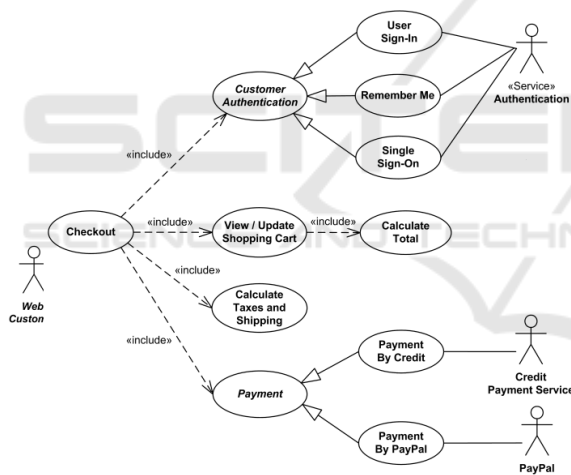


Figure 4: Usecase for Checkout Scenario.

The assessment of reliability is done by translating the FFBD to RBD (Rajput and Chourey, 2015). This provides the reliability contribution of the blocks in attaining desired system reliability. Next step is to annotate the blocks with constraints as the reliability. The added constraint benefit in deriving relevant and specific tests (Petke et al., 2015). There were 2 simulation scenarios created to evaluate reliability Ri and Reliability Importance RIi.

The RBD analysis performed with ReliaSoft BlockSim simulates the system for 500 hours operations. The target reliability in these scenarios is 0.95. The reliability values for 500 hours for each of the blocks obtained are as in figure 7.

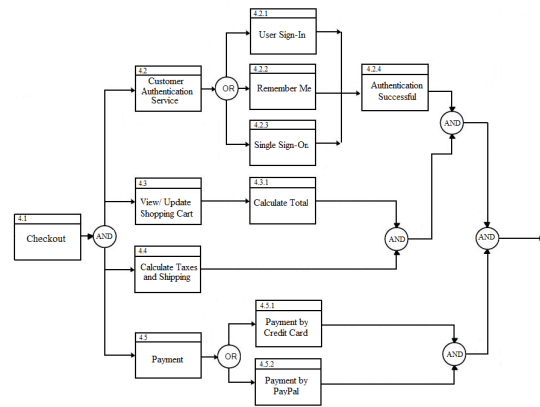


Figure 5: FFBRD for Checkout Scenario.

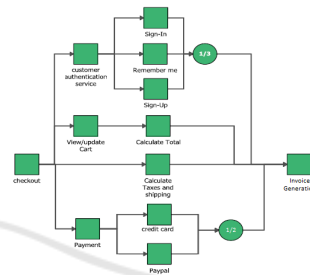


Figure 6: RBD Analysis for Checkout Scenario.

The evaluations suggest the system reliability attained in the scenario 1 is 0.638212 and in the second is 0.641303. If the target reliability is considered, the MTTFF (Hr) calculates to 1151.5 and 1313.71 respectively for the scenarios. The reliability parameter also determines MTBF, Uptime, Downtime, Availability and Maintainability statistics for the system.

General	System Overview for Mission End Time (Hr) = 500			
	Scenario 1		Scenario 2	
Block Name	Reliability	Reliability Importance	Reliability	Reliability Importance
Checkout	0.734223	0.874335	0.698547	0.85361
Customer Authentication Service	0.529059	0.476667	0.647885	0.45236
View / Update Cart	0.848771	0.679685	0.789658	0.667452
Calculate Taxes and Shipping	0.782631	0.903235	0.76610	0.893588
Payment	0.885276	0.978564	0.856412	0.946879
Invoice Generation	0.7267.6	0.824856	0.736589	0.8125477
Sign-In	0.664635	0.601097	0.698751	0.631204
Remember Me	0.485101	0.404569	0.500897	0.56810
Sign-Up	0.529059	0.4417	0.491021	0.375681
Calculate Total	0.664635	0.601097	0.687127	0.61259
Credit Card Payment	0.789564	0.601735	0.713698	0.58123
PayPal Payment	0.814698	0.611855	0.784569	0.596358

Figure 7: Reliability Analysis of Checkout Scenario.

The reliability importance computed in the simulation prioritizes the blocks (Mostafa et al., 2017). The payment, checkout and payment modes are identified as the critical blocks with higher Reliability Im-

portance. These are relevant to the regression testing of system and prioritize the test cases corresponding to the identified blocks.

4 TEST COVERAGE AND TEST SUFFICIENCY

The test case generation may be categorized as a constraint satisfaction problem (Petke et al., 2015; Fraser and Arcuri, 2015). The solution to the problem is approached through control flow graphs. The model elements are converted into an intermediate model to form a directed graph. The graph is marked for entry points, exit points, edges and nodes. The simulated graphs are applied to backtracking algorithm with state space search for solution space. All path set is created and the attribute of the node with maximum reliability becomes the search node. The search node is backtracked and the path for the graph is set as the test case with maximum priority. Similarly, next and subsequent reliable nodes are identified for paths and test cases are organized accordingly. Branching and decision nodes makes subgraphs for the test cases. Thus, the Search Graph (SG) ordered in the reliability value to optimize the test cases is generated (Trivedi and Bobbio, 2017).

Another interesting advantage with the algorithm is that it the path-oriented algorithm chooses the best first search to make the most important test case. The look-ahead and look-back feature of the search algorithm makes the complete test case. The node that are traversed are marked as coloured if the tests have been generated for the graph through that node, with a graph colouring algorithm. Thus the redundancy of tests can be curbed through the algorithm. This prevents the state space explosion of redundant and irrelevant tests being generated through such graph traversal strategies.

4.1 Algorithm for Prioritizing Nodes in the FBGraph

Algorithm for generating Search Graphs (SG): In this step, all possible test sequences are derived from the flow in FFBRD. The sequence of blocks in all independent paths is a path in Search Graph. The guard conditions, OR states, AND states derives the paths in the graph. It may be noted that the graphs generated from the RBDs give maximum coverage as they are path based. The algorithm ensures covering all blocks and all paths from all type of user inputs and flow as responses to the inputs.

```

Begin
    Input: p = path to be traversed
    /the paths of the graphs are traversed
    Output: TCi ,
    testcase of all possible paths p,
    search graphs SG,
    State space (FBi, C, I, F)
    Where FBi is the FunctionBlock
    C is the connections to other Blocks
    I is the initial Block
    F is the final Block or end state
    call all-path-search(FBi) for all connections C
    until look-back(I) and look-ahead(F) is reached
    return SG with start node,
    end node and weights as RI
    // Read the properties of the Block and
    find the maximum value of the reliability Ri
End

```

4.2 Algorithm 2: RTO (Regression Test Optimizer)

Algorithm for optimizing tests: Test optimization here means eliminating the redundant paths generated as test sequences i.e. SG. This also includes execution of graphs in a sequential manner to improve the performance of regression testing. The simulated execution of the transactions in the system generates reliability and reliability importance which prioritizes the test sequences accordingly.

```

Begin
Parameters:
Test Suite TS,
    includes Test Cases TC
    that correlates FB for each cases,
Weights in RI values
//RI is Reliability Importance for a block
for all TCi in TSi (1 ≤ i ≤ n)
    corresponds to each path p in all-path-search
Sort TCi by Ri
Ri: reliability of Block FBi
//reliability of ith block
Rii: Reliability Importance of Block
//reliability importance of ith block
Create reliability-wise-graph path
// sorted on RI
//assigning priority on max(Rii)

TS = Test Scenarios for each SG path

End

```

The algorithm analyses scenarios in the context of component and system reliability. The all path test scenario assures test completeness and test coverage for the regression testing.

5 CONCLUSION AND FUTURE SCOPE

The approach of FFBRD discussed in this paper is an advantageous approach to use models as tools for test generation. This is a preparedness for test during the design phase and keeps the parameters of quality in check. The algorithm saves time and complexity of test generation from other sources. If automation of the process is done, adaptability of the method in processes like Agile will reduce the drawbacks of testing. A rigorous study with the practices of industry and the use of FFBRD in the development process needs to be done. The efficiency of the method and model can thus be validated for quality-based development. Appropriate analysis of requirements can generate test possibility and prioritized requirements. Also, tool support of automating the process can be identified to produce relevant regression test cases. A thorough and systematic research in this direction will make the studies in reliability testing in Software Engineering efficient and productive.

REFERENCES

- Aichernig, B. K., Mostowski, W., Mousavi, M. R., Tappler, M., and Taromirad, M. (2018). Model learning and model-based testing. In *Machine Learning for Dynamic Software Analysis: Potentials and Limits*, pages 74–100. Springer.
- Alanen, M. and Porres, I. (2005). *Model interchange using OMG standards*. IEEE.
- Arora, P. K. and Bhatia, R. (2018). Agent-based regression test case generation using class diagram, use cases and activity diagram. *Procedia Computer Science*, 125:747–753.
- Chourey, V. and Sharma, M. (2015). Component based reliability assessment from uml models. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 772–778. IEEE.
- Chourey, V. and Sharma, M. (2016). Functional flow diagram (ffd): semantics for evolving software. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2193–2199. IEEE.
- Dick, J., Hull, E., and Jackson, K. (2017). *Requirements engineering*. Springer.
- Felderer, M. and Herrmann, A. (2018). Comprehensibility of system models during test design: a controlled experiment comparing uml activity diagrams and state machines. *Software Quality Journal*, pages 1–23.
- Fraser, G. and Arcuri, A. (2015). Achieving scalable mutation-based generation of whole test suites. *Empirical Software Engineering*, 20(3):783–812.
- Gérard, S. Papyrus user guide series: About uml profiling. 2011.
- Gokhale, S. S. and Trivedi, K. S. (2002). Reliability prediction and sensitivity analysis based on software architecture. In *13th International Symposium on Software Reliability Engineering, 2002. Proceedings.*, pages 64–75. IEEE.
- Goševa-Popstojanova, K. and Trivedi, K. S. (2001). Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204.
- Gupta, N., Yadav, V., and Singh, M. (2018). Automated regression test case generation for web application: A survey. *ACM Computing Surveys (CSUR)*, 51(4):87.
- Jena, A. K., Swain, S. K., and Mohapatra, D. P. (2014). A novel approach for test case generation from uml activity diagram. In *Issues and challenges in intelligent computing techniques (ICICT), 2014 international conference on*, pages 621–629. IEEE.
- Kandil, P., Moussa, S., and Badr, N. (2017). Cluster-based test cases prioritization and selection technique for agile regression testing. *Journal of Software: Evolution and Process*, 29(6):e1794.
- Kim, J., Jeong, H., and Lee, E. (2017). Failure history data-based test case prioritization for effective regression test. In *Proceedings of the Symposium on Applied Computing*, pages 1409–1415. ACM.
- Luo, L. (2001). Software testing techniques. *Institute for software research international Carnegie mellon university Pittsburgh, PA*, 15232(1-19):19.
- Lyu, M. R. et al. (1996). *Handbook of software reliability engineering*, volume 222. IEEE computer society press CA.
- McInnes, A. I., Eames, B. K., and Grover, R. (2011). Formalizing functional flow block diagrams using process algebra and metamodels. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(1):34–49.
- Mingsong, C., Xiaokang, Q., and Xuandong, L. (2006). Automatic test case generation for uml activity diagrams. In *Proceedings of the 2006 international workshop on Automation of software test*, pages 2–8. ACM.
- Mostafa, S., Wang, X., and Xie, T. (2017). Perfranker: Prioritization of performance regression tests for collection-intensive software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 23–34. ACM.
- Nelson, W. (1983). *How to analyze reliability data*, volume 6. Asq Pr.
- Nelson, W. B. (2009). *Accelerated testing: statistical models, test plans, and data analysis*, volume 344. John Wiley & Sons.
- Petke, J., Cohen, M. B., Harman, M., and Yoo, S. (2015). Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection. *IEEE Transactions on Software Engineering*, 41(9):901–924.
- Pham, T. and Pham, H. (2017). A generalized software reliability model with stochastic fault-detection rate. *Annals of Operations Research*, pages 1–11.

- Rajput, B. S. and Chourey, V. (2015). Uml based approach for system reliability assessment. *International Journal of Computer Applications*, 131(2):0975–8887.
- Sarma, M. and Mall, R. (2007a). Automatic test case generation from uml models. In *10th International Conference on Information Technology (ICIT 2007)*, pages 196–201. IEEE.
- Sarma, M. and Mall, R. (2007b). Synthesis of system state models. *ACM SIGPLAN Notices*, 42(11):5–14.
- Teixeira, F. A. D. and e Silva, G. B. (2018). Easytest: An approach for automatic test cases generation from uml activity diagrams. In *Information Technology-New Generations*, pages 411–417. Springer.
- Trivedi, K. S. and Bobbio, A. (2017). *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press.
- Wu, L., He, W., Liu, B., Han, X., and Tang, L. (2018). Scenario-based software reliability testing and evaluation of complex information systems. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 73–78. IEEE.
- Yildirim, U., Campean, F., and Williams, H. (2017). Function modeling using the system state flow diagram. *AI EDAM*, 31(4):413–435.
- Yoo, S. and Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120.
- Zheng, Z., Trivedi, K. S., Qiu, K., and Xia, R. (2017). Semi-markov models of composite web services for their performance, reliability and bottlenecks. *IEEE Transactions on Services Computing*, 10(3):448–460.