2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
Banff Center, Banff, Canada, October 5-8, 2017

Analytics-based Safety Monitoring and Verification

Hadi Hemmati
Department of Electrical & Computer Engineering
University of Calgary
Calgary, AB, Canada
hadi.hemmati@ucalgary.ca

Syed S. Arefin, Taha R. Siddiqui
Department of Computer Science
University of Manitoba
Winnipeg, MB, Canada
{ssarefin, trsid}@cs.umanitoba.ca

Abstract— **Safety-critical systems in domains such as aviation, railway, and automotive are often subject to a formal process of safety certification. The goal of this process is to ensure that these systems will operate safely without posing risks to the user, the public, or the environment [1]. It is typically expensive and time consuming for companies to certify their software. Therefore, any attempt to automate any part of the required process is very appreciated. In this research project, we report on our on-going project with an industry partner in the avionics domain, where we propose a framework to combine specification mining, model-based testing, and analytics to help with monitoring and verification of safety critical systems.**

Keywords—specification mining; model-based testing; data analytics; satefy crtical systems; testing, debugging.

I. Introduction

Avionics software systems are safety-critical software embedded in an aircraft hardware that control and monitor the aircraft. Unmanned Aerial Vehicle (UAV) is an avionics system with no pilot on board and usually, flown by a pilot at a ground control station. A UAV can also fly autonomously based on predefined flight plans using a software called Autopilot. The Autopilot helps automate the process of controlling and guiding the aircraft [2].

Aviation software industry observes one of the highest standards of safety control, where their embedded software systems go through a set of rigorous standard checks, before entering to the commercial market. An autopilot software of an Unmanned Aerial Vehicle (UAV), is an example of an avionics software systems, which must be certified in certain UAV application domains.

In this project, we collaborate with a word-leader in building autopilot systems for commercial UAVs, to provide them with novel tools and mechanisms that help automating several tasks required by standards such as DO-178C [3]. One of the major demands of DO-178C is having a set of explicitly written "software requirements" for the safety critical software and a set of test cases that verifies those requirements, plus a mapping between each requirement and its test cases to provide a two-way traceability between the requirements and the tests.

To achieve such demands, the high-level goal of this study is to provide a set of state-of-the-art techniques and tools to improve the current practice of requirement generation, program comprehension, monitoring, testing, and debugging for safety-critical software systems, in general, and for our industry partner, in particular. The study consists of four concrete objectives as follows:

- O1: Providing a semi-automated technique to help with (safety) requirement generation and traceability from requirements to test cases
- O2: Automate test generation from specification and provide a traceability from test cases and requirements
- O3: Prioritize test cases based on their historical failures and different coverage criteria
- O4: Providing a semi-automated technique to help with fault localization and debugging.

Technically speaking, a set of techniques from "model-driven engineering" and "data analytics" will be used to realize the above objectives, in the context of two sub-projects as follows:

- **Project 1 – An Interactive Specification Mining for Requirement Generation, System monitoring, and Debugging:** In this project, we propose a semi-automated specification mining technique that is tailored for our context. The technique resulted in an interactive tool that not only help developers with program comprehension and requirement generation, but also speeds up the debugging process. This project will realize O1 and O4.

- **Project 2 – Model-based Test Generation and Prioritization:** In this project, we use a combination of model-based testing and data analytics to automatically generate test cases that assure high specification-level coverage. We also use data analytics techniques to predict failure probability of test cases and prioritize them based on such probabilities. This project will realize O2 and 3.

This is an on-going project. Some of the techniques and tools proposed above are already implemented and are being evaluated in the company, by controlled experiments and interviews. In the rest of this paper, we will explain some detailed about the proposed tools and techniques and share some initial feedbacks.

The rest of this paper, is organized as follows: In Section II, background and related work will be explained. Section III details the proposed techniques and tools. Section IV explains the current state of the project and its future directions. Finally, Section V, summarizes the paper.

II. BACKGROUND AND RELATED WORK

In this section, the two most relevant topics to this project (model-based testing and specification mining) and some related work will be explained.

A. *Model-based testing*

Model-based Testing (MBT) is an efficient automated test generation process [4, 5]. This technique uses the model of the system requirement and functionality as a basis to generate test cases. It is an application of model-driven engineering for software verification and testing.

The goal of the MBT is to automatically generate executable test cases based on the specification model. These models represent functionality but also can include performance, safety, and security concerns. The MBT process is composed of the following steps represented in Figure 1.

1) The test designer manually models the system under test (and if required its environment). While modeling the system, the designer may focus only on specific parts (e.g., components, classes, features, etc.) of the system or specific aspects (e.g., functionality, safety, security, performance, etc.) of the system, (s)he is interested to test. The narrow focus typically increases the scalability of the technique.

2) The MBT tool automatically generates the abstract tests from the model. To do that the model, typically, is converted to a graph of some form. Then a model coverage strategy (a graph traversal algorithm) is applied on that graph to create a set of test paths (sequence of nodes and edges). These abstract test cases identify the scenarios that are going to be verified.

3) The abstract test cases are too generic and need language-specific data to be executable. In addition, to generate executable tests, the MBT tool needs to add specific input data values for each method call in a test path (method calls that are invoked along the execution of the scenario under test). The test data can be generated randomly or using a more sophisticated algorithm (e.g., evolutionary search, symbolic execution, etc. [6]).

4) Finally, the generated test cases are executed (typically within a test execution framework) and the outputs are analyzed and reported. This requires the MBT tool to create test assertions depending on the expected state of the system, to compare the actual behavior with requirement of the system (represented as the specification models).

There are several applications of Model-driven Engineering in safety-critical avionics systems; Model-based testing is only one of them. We have studied several UML-based solutions [7—11] that can be attributed to capture safety-related information in UML models, in case they can be a useful add-on to a typical UML model.

For example, a study conducted by Stallbaum et al. [12] introduced a new UML profile that can be used to extend standard UML test models for MBT with information that is relevant to airworthiness certification. Those test models can serve as supporting artifacts of certification in case MBT is applied. We found that none of the solutions proposed in this category of studies used an MBT technique to automate system level test generation. Instead, they only focused on modeling the system along with the safety-related information.

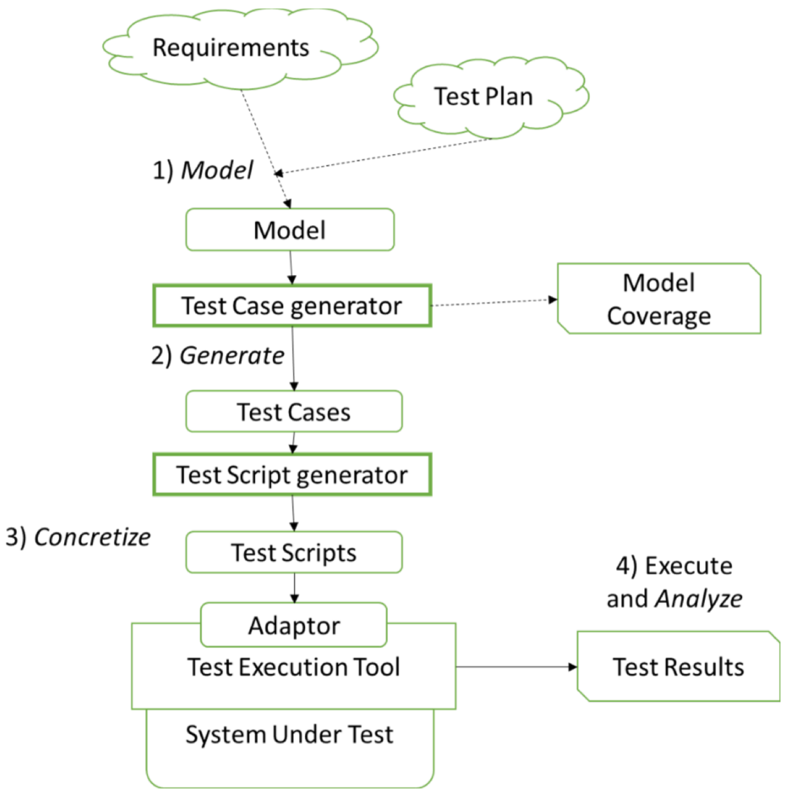

Figure 1 Steps of Model-based Testing (MBT) Technique

In our study, we only use some basic concepts such as timing from real-time domain which can still be modeled in UML standard diagrams. So we do not use any extra profile.

B. *Specification mining*

Specification Mining [13] is a relatively new research area in software engineering. It can be seen as an application of data mining on software engineering datasets. In the past, several research projects have targeted the problem of specification mining as reverse engineering [14—16]. The two major approaches for software reverse engineering are static and dynamic analysis.

Briefly described, static analysis uses the source code or other artifacts as is, without execution. On the other hand, dynamic analysis approach works by executing the real code to get execution traces and then mining specifications from them.

It requires instrumenting the source code to get logs from real execution in the form of traces. Execution traces typically consist of sequences of method calls, and other related information. These sequences can be generated by instrumenting the program and running the system with different inputs (different scenarios), the more the better, to cover the overall behavior of the system and hence producing correct and valid specification of the system.

Higher coverage of the test inputs generates more accurate and complete specification models. It not only helps modelling behavior of a software system, but is also extremely useful for a wide range of software engineering tasks, such as software validation and verification, anomaly detection [17], test case generation [18], etc.

III. PROPOSED METHODOLOGY

The overall goal of this proposal is helping safety critical software companies with getting certifications such as DO-178C. The focused areas of the project are on verification and monitoring. Figure 2 illustrates the high-level idea of this proposal.

The following six steps are involved in this methodology:

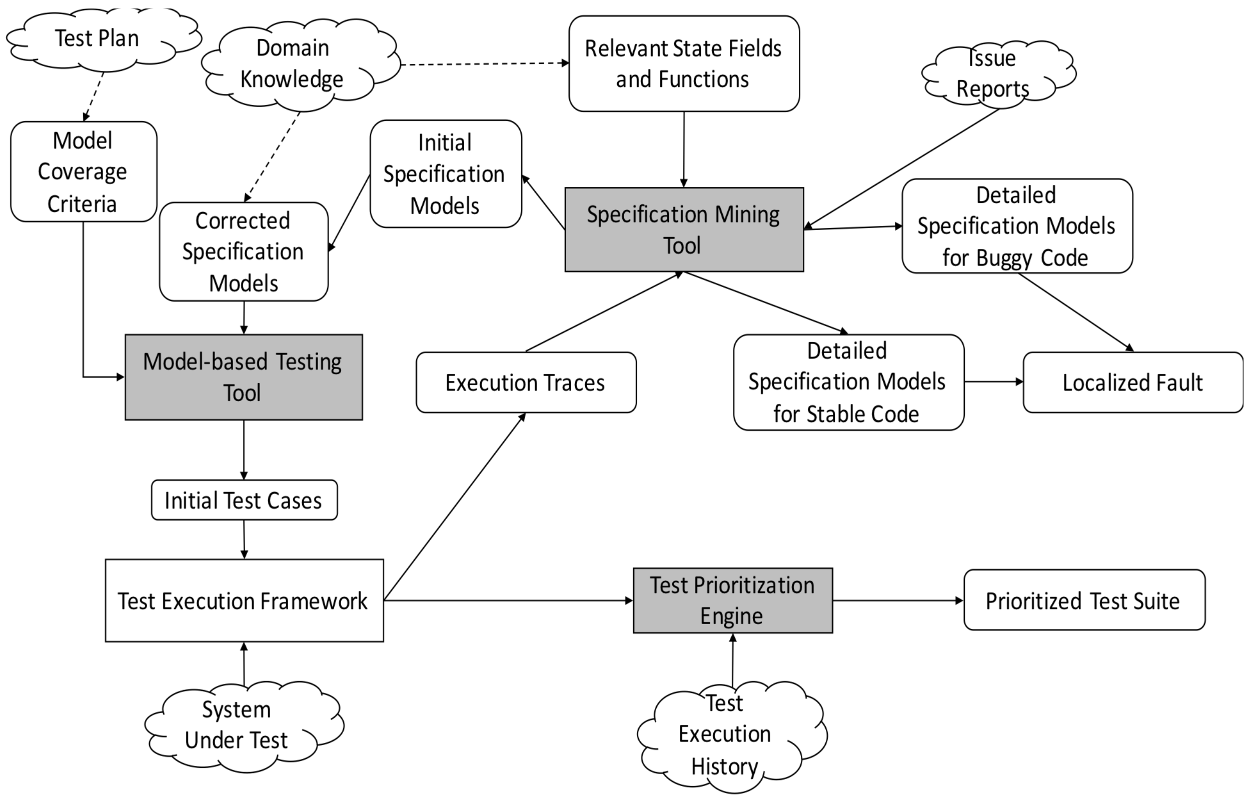
Test Plan
Domain Knowledge
Relevant State Fields and Functions
Issue Reports
Model Coverage Criteria
Initial Specification Models
Corrected Specification Models
Specification Mining Tool
Detailed Specification Models for Buggy Code
Model-based Testing Tool
Execution Traces
Detailed Specification Models for Stable Code
Localized Fault
Initial Test Cases
Test Execution Framework
Test Prioritization Engine
Prioritized Test Suite
System Under Test
Test Execution History

Figure 2. The combined specification mining and model-based testing strategy

IV. PROPOSED METHODOLOGY

The overall goal of this proposal is helping safety critical software companies with getting certifications such as DO-178C. The focused areas of the project are on verification and monitoring. Figure 2 illustrates the high-level idea of this proposal.

The following six steps are involved in this methodology:

1) Seting up the abstraction-level and verification aspects

At this first step, the domain expert decides what aspects of the system should be monitored and verified. The experts define this by selecting a set of state variables from a given list. These variables define what states should be extracted at runtime.

2) Mining the initial specification models

The interactive specification mining tool will generate a state machine representation of the system under test, by running the system with a set of initial test cases. The tool logs the execution traces using a profiler tool and finally abstracts the execution traces into state machines, where the states are identified by the important changes in the selected state fields.

3) Model validation and augmention

In this step, the engineers need to validate the state machines and define a test strategy (e.g. all node coverage). They also have to create the explicit requirements based on the mined behavior and map the requirements with paths or states in the state machine.

Note that though this step is manual but the fact that the engineer/developer/analyst creates requirements and safety cases by looking at the high-level models rather than low-level code base already helps speed up process and might improve the precision as well.

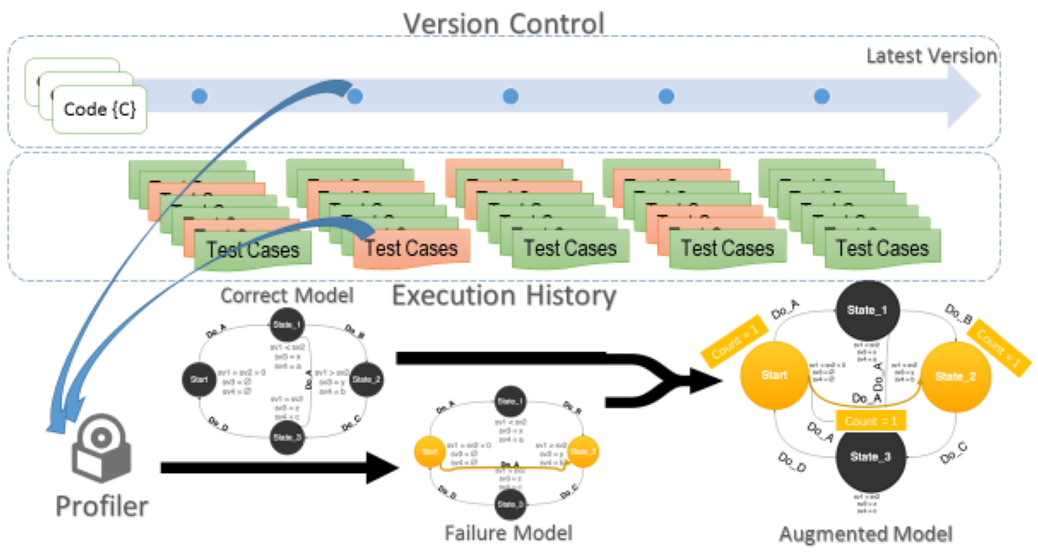

Figure 3. Overview of Fault Augmentation

1) Seting up the abstraction-level and verification aspects

At this first step, the domain expert decides what aspects of the system should be monitored and verified..

The experts define this by selecting a set of state variables from a given list. These variables define what states should be extracted at runtime

2) Model-based test generation

Beside helping with requirement generation, the generated models from step 3 are also used for automatic test generation [19]. A model-based testing tool will get the models as input and generates executable test cases as output. It can also measure code coverage and other relevant metrics.

3) Monitoring and debuging aids

In this step, the validated models of step 3 will be fed back to the specification mining tool to extract low-level state machines for debugging purpose.

Basically, for any reported issue, the QA team can run the specification mining tool for the relevant test cases and get as detailed as desired state machine (again using the interactive abstraction set up, explained in step 1). Such state machines can also be generated for older versions of the same feature. Then the QA engineer will compare the two state machines to root cause and debug the issue.

4) Analytics-based moedel augmention and test prioritization

The framework will also keep track of all test failures and issues [20, 21]. This information can later on be used for predicting which components of the system are more error-prone and high-light the test cases that are more likely to fail. So that in cases where the testing budget is restricted, one can only execute the most important tests. This information can also be integrated back into the model so that the engineers can refine their test plans and safety cases. Figure 3 summarizes this step.

V. Current Status and Future Directions

In collaboration with our industry partner, we already have implemented the model-based testing as well as the specification mining tools. At this current moment, we are evaluating the specification mining tool with a series of interviews with the developers at the company.

The next step will be evaluating the model-based testing tool and then combining the two tools into one framework. Finally, we need to add the analytics engine for augmenting the fault information.

The initial feedback from industry was quite promising and encouraging to continue in this direction.

Though we have applied this idea on a software-centric system, but the overall strategy can be applied on any embedded system with the ability on logging the execution events [22, 23]. This will particularly be useful for monitoring and debugging of any safety-critical system.

VI. Summary

This paper reports on an on-going project where specification mining, model-based testing, and data analytics techniques are used for generating and augmenting specification models of a system under test. These models are then used for test generation. The overall, idea is that the proposed semi-automated approach for behavior extraction will help the QA engineers to build a functional model of their system and makes the representation of safety goals more accurate but also faster. In addition, the automated test generation engine will verify those functionalities and/or safety cases. The current project is tailored for a specific embedded software, but can be generalized to be applicable for monitoring and verification of most safety-critical systems.

References

[1] S. Nair, J. L. De la vara, M. Sabetzadeh, L. Briand, "An extended systematic literature review on provision of evidence for safety certification", *Information and Software Technology*, vol: 56, no: 7 689-717, 2014.

[2] Automated Flight Controls. www.FAA.gov. Federal Aviation Administration. (Retrieved 19 August 2016).

[3] DO-178C, Software Considerations in Airborne Systems and Equipment Certification. Washington, D.C., USA, December 2011.

[4] J. Offutt, and A. Abdurazik. "Generating tests from UML specifications", International Conference on the Unified Modeling Language. Springer Berlin Heidelberg, 1999

[5] H. Hemmati, A. Arcuri, and L. Briand. "Achieving scalable model-based testing through test case diversity", *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22.1 (2013): 6.

[6] S. Ali, L. Briand, H. Hemmati, R.K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation", *IEEE Transactions on Software Engineering* 36.6 (2010): 742-762.

[7] G. Zoughbi, L. Briand, and Y. Labiche, "Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile", Software & Systems Modeling 10.3 (2011): 337-367.

[8] G. Zoughbi, L. Briand and Y. Labiche, "A UML Profile For Developing AirworthinessCompliant (RTCA DO-178B) Safety-Critical Software," Carleton University, Technical Report SCE-05-19, December, 2006.

[9] M.A. de Miguel, J.F. Briones, J.P. Silva, and A. Alonso, "Integration of safety analysis in model-driven software development," IET Software, vol. 2, no. 3, Jun. 2008.

[10] K.T. Hansen and I. Gullesen, "Utilizing UML and Patterns for Safety Critical Systems," Proc. Workshop on Critical Systems Development with UML, in conjunction with the International Conference on the UML, 2002.

[11] J. Jürjens, "Developing Safety-Critical Systems with UML," Proc. International Conference on the UML, LNCS 2863, pp. 360-372, 2003.

[12] H. Stallbaum, and M. Rzepka, "Toward DO-178B-compliant Test Models." Model-Driven Engineering", Verification, and Validation (MoDeVVa), 2010 Workshop on. IEEE, 2010.

[13] C. Liu, ed. "Mining software specifications: Methodologies and applications", CRC Press, 2011.

[14] T. Ziadi, et al., "Towards a language-independent approach for reverse-engineering of software product lines", Proceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, 2014.

[15] H. Bruneliere, et al., "Modisco: A model driven reverse engineering framework", Information and Software Technology 56.8 (2014): 1012-1032.

[16] S. Hassan, et al., "Software Reverse Engineering to Requirement Engineering for Evolution of Legacy System", IT Convergence and Security (ICITCS), 2015 5th International Conference on. IEEE, 2015.

[17] A. Valdes, K. Skinner, "Adaptive, model-based monitoring for cyber-attack detection", In: Recent Advances in Intrusion Detection, Springer, pp 80–93, 2000.

[18] R. Taylor, M. Hall, K. Bogdanov, J. Derrick, "Using behavior inference to optimize regression test sets". In: Testing Software and Systems (ICTSS'12), Springer, pp 184–19, 2012.

[19] A., Shaukat, H. Hemmati, "Model-based testing of video conferencing systems: challenges, lessons learnt, and results", Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on. IEEE, 2014.

[20] T. Bin Noor, H. Hemmati, "Test case analytics: Mining test case traces to improve risk-driven testing", *Software Analytics (SWAN), 2015 IEEE 1st International Workshop on*. IEEE, 2015.

[21] T. Bin Noor, H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data", *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*. IEEE, 2015.

[22] W. Shang, et al. "Assisting developers of big data analytics applications when deploying on hadoop clouds", Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.

[23] H. Malik, H. Hemmati, and A.E. Hassan, "Automatic detection of performance deviations in the load testing of large scale systems", Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.