Quality Assurance for Component-based Systems in Embedded Environments

Wenbin Li, Franck Le Gall
Easy Global Market
Sophia Antipolis, France
firstname.lastname@eglobalmark.com

Panagiotis Vlacheas
WINGS ICT Solutions
Athens, Greece
panvlah@wings-ict-solutions.eu

Alexey Cheptsov
High Performance Computing Center
Stuttgart, Germany
cheptsov@hlrs.de

***Abstract*—Quality assurance for embedded systems is challenging to the heterogeneity, connectivity and constructivity of environments. In this paper, we present the quality assurance design and implementation driven by model-based testing (MBT) for component-based applications in embedded environments. The work is conducted within the EU H2020 project PHANTOM, and the quality assurance in PHANTOM consists of two stages, i.e., early validation and test execution, for both functional and non-functional verification and validation. Early validation stage is carried out in parallel with application development without executing applications to eliminate early design defects, while test execution stages involves functional and non-functional testing, which start in parallel with the application development and end with the execution of test cases against systems under test. All activities are applied to three industry use cases upon hardware agnostic platforms brought by PHANTOM technology, and the results show that combining early validation and test execution enables early and thorough defect detections all along development lifecycle and improve the efficiency and effectiveness of quality assurance.**

Keywords—quality assurance, model-based testing, early testing, component-based, embedded environment, verification, validation

I. INTRODUCTION

The future of Quality Assurance (QA) expects intelligence and automation to meet the requirements of speed, efficiency and coverage [1]. With the development of Model-Based Design [2] technologies, Model-Based Testing (MBT) is a popular way to drive QA of software by involving models to formalize and automate as many testing activities as possible and thereby to increase both effectiveness and efficiency. MBT designates any kind of "testing based on or involving models"; MBT models represent the system under test (SUT), its environment, or the test itself, which directly supports quality assurance activities such as planning, control, analysis, implementation, execution and reporting [3]. MBT is being increasingly adopted to automate testing and brings intelligence to quality assurance by combining model-based design and technologies.

One existing challenge of QA is to ensure the quality of embedded systems due to the heterogeneity, connectivity and constructivity of embedded environments [4]. More efforts are expected to study how different technologies can be applied to QA in embedded environments. A number of MBT approaches have been proposed to drive the study [5] and MBT tools have been developed covering testing steps and processes [6]. In existing work, MBT is applied in industry for real case scenarios [7] with a special focus stage; furthermore, MBT also promises early testing activities for fault detection [8] but still involves the execution of early prototypes. Since the test execution against prototypes comes after the prototype development, the early testing is limited to the end of prototype stage. A complete QA solution to cover different software development stages is required. To conduct quality assurance activities in an earlier stage, relevant activities requires pre-analysis of the intended implementation without real execution of applications, and this is especially important in embedded environments to ensure that applications are validated before deployment.

In this paper, we present the quality assurance activities driven by MBT in the H2020 PHANTOM Project [9], in which MBT is designed and applied to drive the functional and non-functional verification and validation for component-based applications in embedded environments. PHANTOM project aims at providing a cross-layer and multi-objective programming approach for next generation heterogeneous parallel computing systems, along with a platform for application optimization, monitoring, security and testing. The targeted embedded environments include CPU, GPU and FPGA with three industrial cases respectively in the domains of Telecom Network Management, Satellite Surveillance and Airflow Simulation. All use case applications are composed by of a number of components as networks. The objective of quality assurance in PHANTOM is to verify and validate functional and non-functional properties for use case applications as well as individual components upon PHANTOM platform. Besides thorough and systematic requirements, one particular requirement is the early application verification which should enable developers to verify their applications very early in the life cycle, even in parallel of the development. In this paper, we present the design and implementation of QA driven by MBT in PHANTOM as two stages, i.e.., early validation and test execution. Early validation checks the intended functionalities of the intended implementations and estimates the performance in parallel with the application development, while test execution provides and executes concrete tests to conduct thorough functional and non-functional testing.

The remainder of the paper is organized as follows: section 2 presents a general view of the PHANTOM project; section 3 introduces the design of QA driven MBT in PHANTOM, and section 4 describes the details; section 5 illustrates the implementation and results; at last section 6 concludes our contributions.

II. PHANTOM OVERVIEW

PHANTOM project aims at delivering an integrated cross-layer (hardware and system software/programming environment) multi-objective and cross-application approach that will enable next generation heterogeneous, parallel and low-power computing systems, while hiding the complexity

978-1-5386-9131-1/18/$31.00 ©2018 IEEE

of computing hardware from the programmer, thus fostering productivity in programming. The architecture of PHANTOM platform is shown in Fig. 1 [10].

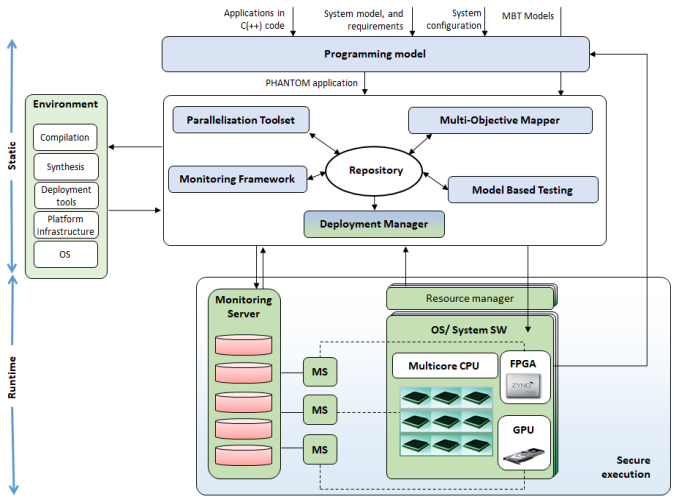


Fig. 1. PHANTOM Architecture

From the top of the figure, the Programming Model leverages a functionally-oriented component-based methodology to the application design, by use of which the developer structures their application as an interconnected network of communicating components. In source code based on programming model, both functional requirements (e.g., HWs to use) and non-functional requirements (e.g., expected performances) are also captured. The Repository serves as an intermediate system with unified interfaces for data exchange among PHANTOM components. The Parallelization Toolset optimizes the code and provide all necessary information through annotated code (e.g., suited libraries, loop information) that will be used as guidelines by the Deployment Manager. The Deployment Manager finalizes the code with communications and consistency, and then builds different components on the selected HWs specified by the optimized deployment plan. The Multi-Objective Mapper provides the optimized deployment plan for PHANTOM applications which indicates the mapping of the parallelizable components on the target HWs (i.e., CPU, GPU and FPGA). During execution, the Monitoring Framework monitors both the application and the hardware infrastructure to collect, store and analyze performance-related data. Security mechanisms based on access control and execution integrity ensure the secure of execution within PHANTOM environments. All along the static and runtime stage illustrated in the figure, MBT ensures the functional and non-functional properties of applications deployed upon PHANTOM platform.

Based on this architecture, PHANTOM is able to provide use case applications with a hardware-agnostic software platform over reconfigurable multi-core and heterogeneous HWs. The software platform offers the means for programing, deployment, multi-dimensional optimization, monitoring, security and QA.

III. Quality Assurance Design

We have designed four activities driven by MBT (1 to 4 in Fig. 2) in PHANTOM, which lie in two main stages (I and II in Fig. 2) for SUT quality assurance. Fig. 2 lists the QA activities and the alignment with software stages.

Early validation is the first QA stage in PHANTOM, which tests and validates the design of an application's functional and non-functional properties. Early validation only relies on design specifications but does not require the execution of applications, and it is a stage achieved in parallel with the application development to detect design defects for follow-up correction and prevent them from escalating to implementation.

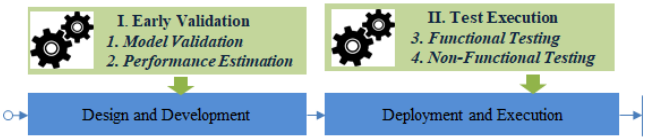


Fig. 2. QA Activities driven by MBT

Specifically, the following two activities - model validation and performance estimation - are conducted during early validation. During model validation, MBT models are created following design specifications of applications and thus are able to reflect both static representation and dynamic behavior of SUTs. Then the models can be simulated by tools to detect potential design flows; performance estimation covers the application's non-functional aspect by calculating the performance of a newly design applications following predefined estimation patterns.

Test execution is the second QA stage in PHANTOM to execute concrete test cases generated from MBT models against the SUT and collects thorough testing results for both functional and non-functional properties. Testing results are sent back to developers for further analysis and improvement. During the whole test execution stage, traceability is kept between SUT specifications, test cases and testing results, so that developers can easily trace the defects sources from testing results to design specifications, and correct the defects based on the testing results. Both of the two activities in this stage, i.e., functional testing and non-functional testing, shares the same workflow with different testing aspects. Correspondently, the MBT models and the executed test cases for the two activities are different from one another.

IV. Quality Assurance Activities

In this section, we present the detailed design of each QA activity.

A. Model Validation

The model validation simulates the MBT models to check if the intended implementation contains any functional defects such as deadlock or over-designing (parts of the model never activated) and provides a summary for all detected functional defects. The corresponding workflow is illustrated in Fig. 3. In all figures in this section, the green rectangles represent steps in one activity. The start and end point of an arrow represents inputs and outputs of this step, and the start and end point of the whole workflow is the global inputs and outputs of this MBT activity.

Step 1. Model Creation. In the first step, we create MBT models from use case specifications. The specifications define the testing requirements or the aspects to test of SUT (e.g., functions, behaviours and performances). The created MBT models represent high-level abstractions of SUT and are described by formal languages or notations such as UML,

PetriNet and BMPN. In PHANTOM, communicating state machine is used as the meta model for MBT models to consider the communication of application components.

- Inputs: Design Specifications
- Outputs: MBT Models

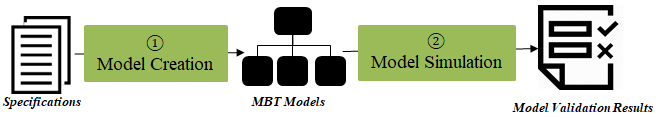


Fig. 3. Model Validation Workflow

Step 2. Model Simulation. In this step, simulation scenarios are automatically generated from MBT models by tools and are used to simulate MBT models. During model simulation, MBT models are validated regarding whether, under what conditions, and in which ways a part of model could fail to produce the correct outputs, and the corresponding deadlocks and over-designing are recorded. The model validation results are analyzed to correct the functional design defects before implementation.

- Inputs: MBT models
- Outputs: Model validation results

B. *Performance Estimation*

Performance estimation estimates the non-functional properties (e.g., execution time, energy consumption) of newly designed applications by considering PHANTOM component network and previous non-functional testing results. PHANTOM applications are composed of components, and a new application can be easily created to recompose existing components in a different way for different tasks. If all the inner components of the new application are previously tested with performance results, the performance of the new application is then deduced by analyzing their composition patterns and previous testing results. The corresponding workflow is illustrated in Fig. 4.

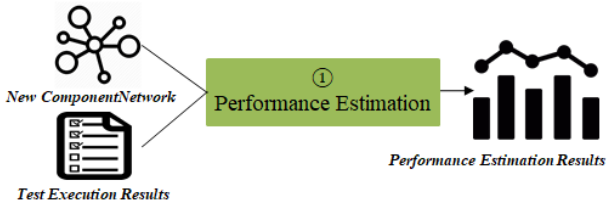


Fig. 4. Performance estimation workflow

Step 1. Performance Estimation. Each application is described by an xml file of component network indicating the internal components of an application and the patterns how the components are composed together. When given a component network description of an application, this step identifies the composition patterns among inner components, estimates the performance of the application based on an estimation model related to each non-functional property as shown in TABLE I, and then provides the estimation results.

- Inputs: Component network description and previous test execution results
- Outputs: Performance estimation results

TABLE I. TABLE I. ESTIMATION MODEL

Property Name	Estimation Models for Composition Patterns			
	Sequence	Parallel	Condition	Iteration
Execution Time (ET)	$\sum_{i=1}^{n} et_i$	$\max(et_i)$	$\max(et_i)$	(et)*k
RAM Usage (RU)	$\sum_{i=1}^{n} ru_i$	$\sum_{i=1}^{n} ru_i$	$\max(ru_i)$	(ru)*k
Reliability (RE)	$\prod_{i=1}^{n} re_i$	$\prod_{i=1}^{n} re_i$	$\min(re_i)$	$(re)^k$
Energy Consumption (EC)	$\sum_{i=1}^{n} ec_i$	$\sum_{i=1}^{n} ec_i$	$\max(ec_i)$	(ec)*k

We have identified four types of composition patterns (i.e., sequence, parallel, condition and iteration) to represent the composition relations among application components. They cover most of the structures specified by workflow patterns [11]. For each performance property, we have defined an estimation model to calculate the application's performance based on their individual components' performance and composition patterns.

This is particularly useful to estimate the performance of applications that reuse existing components and compose them in different ways to complete other computing tasks, such as airflow simulation use case to simulate different topological structures with same components.

C. *Functional and Non-functional Testing*

Test execution is the second MBT stage in PHANTOM to execute concrete test cases against the SUT and collects thorough testing results for both functional and non-functional properties. Driven by MBT, the test cases are automatically generated by tools from MBT models; comparing to the manual preparation of test cases, MBT is able to provide fast test generation and systematic test coverage.

The corresponding workflow of functional and non-functional testing activities is illustrated in Fig. 5. Both functional testing and non-functional testing shares the same general workflow. However, since the functional testing and non-functional testing have different testing aspects, the MBT models and the executed test cases for the two activities are different from one another. The MBT models and test cases of functional testing focuses the inputs/output data flow for functionality, while the ones for non-functional testing collects performance information during the test execution and checks if the SUT meets the performance criteria.

Step 1. Model Creation. In the first step, we create MBT models from use case specifications for test generation purpose.

- Inputs: Specifications
- Outputs: MBT models

Step 2. Test Generation. In the second step, tools automatically generate abstract test cases from MBT models when applying the test selection criteria. Test selection criteria guide the generation process by indicating the interesting focus to test, such as certain functions of the SUT or the structure of the MBT model (e.g. state coverage, transition coverage and data flow coverage). This process is

automated by tools with test selection criteria options corresponding testing requirements.

- Input: MBT models
- Output: abstract test cases

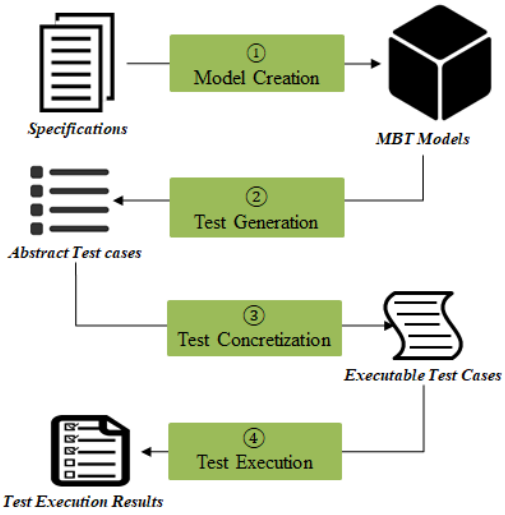


Fig. 5 Functional and non-functional testing workflow

Step 3. Concretization of Test Cases. The third step concretizes the abstract test cases from step 2 to executable test cases with mappings between the abstraction in MBT models and system implementation details. Executable test cases contain low-level implementation details and can be directly executed against the SUT.

- Input: abstract test cases
- Output: concrete test cases

Step 4. Execution of Test Cases. The executable test cases are automatically executed against the SUT. During the execution, the SUT is provided with inputs from each test case, and the outputs (for functional testing) and performance information (for non-functional testing) are collected to generate test verdicts.

- Input: executable test cases, system adapters
- Output: test verdicts, non-functional information

The four steps are performed iteratively and incrementally throughout development, helping to guarantee full alignment with the test objectives and to keep MBT modelling activities efficient. The MBT process thus starts in parallel with the application development and assists the developers through the entire development process.

V. Use Cases and Implemntation

We have implemented all QA activities for the three use case applications in PHANTOM project.

The use case of Telecom Network Management is based on a microwave radio bridge product family that exploits the characteristics and the peculiarities of the embedded, hard real-time systems class. The proposed system is a commercial product representative of the complexity of the telecommunication equipment, but small enough to stress and validate specific characteristics of the PHANTOM platform under a well-controlled environment.

The use case of Satellite Surveillance is an operational solution developed to provide added-value support to maritime situational awareness via Earth Observation (EO) technologies. The application receives satellite images, as well as other information from multiple sources, to not only detect ships but also categorize the detected ships. The results are then presented in a user web interface. The focus of the tool is on the detection of small ships, in order to prevent cross-border crime, irregular migration, and to contribute to protecting migrants' lives.

The use case of Airflow simulation is built around a Parametric Study for complex simulation scenarios as the Computational Fluid Dynamics (CFD), which is a large class of HPC applications. Parametric study usually aims to evaluate the reactive behavior of the investigated physical system when varying the configuration of the input parameter set. The CFD parametric study involves thousands of possible configurations that must be evaluated. Therefore, it is a massively parallel application that will be able to take substantial advantage from running on heterogeneous infrastructure in terms of the individual performance but also in terms of the overall execution time for all studies.

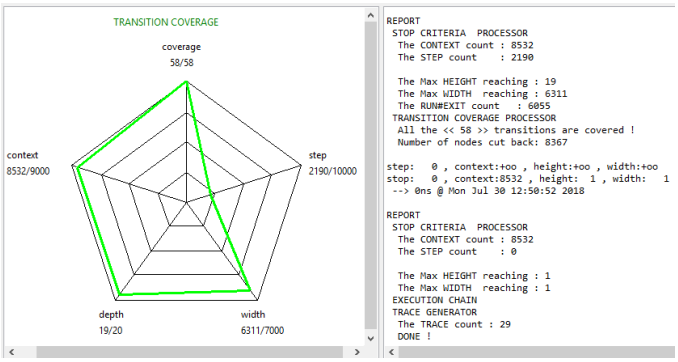
Regarding the implementation details, for model validation, we have developed the MBT models for each use case application following communicating state machine and simulated the MBT models by DIVERSITY tool [12] to detect deadlock or over designing. The MBT models based on communicative state machine are created in xLIA (eXecutable Language for Interaction & Assemblage) language [12], due to the variety of primitives and the support of encoding all classical semantics. We use DIVERSITY for model simulation purpose due to its support for symbolic execution. Symbolic execution uses symbolic parameters to represent simulation inputs rather than concrete numerical values so that multiple scenarios can be evaluated at the same time to simulate the models and explore the model behaviors more efficiently.

For Performance Estimation, we have developed an MBT module, i.e., model-based estimator, to take as inputs of the application's component network description and previous testing results to produce performance estimation results following our estimation model presented in previous section.

For test execution, we have developed a number of MBT modules for the test execution stage to enable functional and non-functional testing. We have developed MBT models for each use case based on communicative state machine in xLIA language and generated test cases from MBT models by applying selection criteria (i.e. exploration, transition coverage and behaviour selection). Test generation from MBT models is achieved by DIVERSITY. DIVERSITY takes communicative state machine defined in xLIA as input and generates test cases in TTCN-3 [13] following one of the three selection criteria. TTCN-3 is a standardized testing language developed by ETSI (European Telecommunication Standards Institute), and we use TTCN-3 language to describe the test cases due to the multiple testing purpose support (e.g. real-time support, distributed testing support) and performance reasons. We have also developed the codecs/decodecs to improve and concretize the generated test cases with real testing data and the system adapters to enable test execution. The development of models and other testing modules starts in parallel with development based on

specifications, and once the application is ready to run, concrete test cases are executed to conduct both functional testing and non-functional testing. TITAN [14] is used to for the execution of test cases, which a TTCN-3 compilation and execution environment with an Eclipse-based IDE supporting test execution as well as result reporting.

Based on the implementation, we have conducted the four QA activities in PHANTOM in Ubuntu 14.04 environment. For illustration purpose, Fig. 6 illustrate the model validation and performance estimation results for satellite surveillance use case; Fig. 7 presents the functional testing results of the telecom network management use case; Fig. 8 shows the non-functional testing results for the airflow simulation use case.



Input	*Deployment Plan*	*Estimation*	
		ET (ms)	*EC (watt)*
component_network_1.xml	*deployment_plan_1.xml*	*700*	*0.48*
component_network_2.xml	*deployment_plan_2.xml*	*1148*	*0.72*
component_network_3.xml	*deployment_plan_3.xml*	*3260*	*1.24*

Fig. 6 Model validation and performance estimation results

The top part of Fig. 6 shows the model validation result from DIVERSITY indicating no over-designing or deadlock in the MBT model, and the bottom part shows the performance estimation results (i.e., execution time and energy consumption) for the component network files of three applications with specific deployment plans.

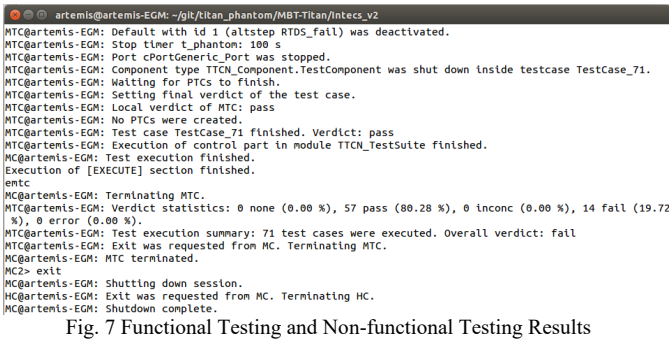


Fig. 7 Functional Testing and Non-functional Testing Results

From Fig. 7, we can see that the execution result of 71 functional test cases by TITAN, in which 57 test cases pass and 14 fail.

Fig. 8 shows the non-functional testing result of the execution of the same test case under 100 different configurations in one airflow simulation scenario. Each configuration provides a different value for the simulation parameter to study the simulation steps that a simulation algorithm takes. The x axis represents the value of the parameter h; the y axis represents the total simulation steps when given different h value.

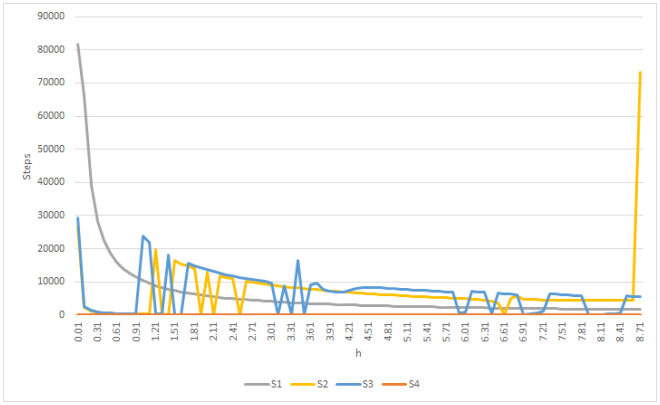


Fig. 8 Non-functional Testing Results

Through the whole QA process in PHANTOM, the feedback from industry use case partners is that the activities largely improve the efficiency and testing coverage for the three applied domains. Especially, early validation stage is able to eliminates the functional and non-functional flaws during design and prevent the escalation to code level. The test execution automates the test generation and execution which provide systematic testing coverage and efficient test execution support.

VI. Conclusion

The verification and validation of embedded systems is challenging but important in software lifecycle for quality assurance. In this paper, we present an end-to-end QA solution driven by MBT with four activities and apply it to three industrial applications in embedded and parallel environments. The work is able to improve both QA effectiveness and efficiency in embedded environments, while remaining general enough to be reused to test component-based applications when internal components and their composition structures are specified.

As a future work, we are currently exploiting how the testing execution stage can be combine with online testing strategy to generate and execute adaptive test cases on the fly; we also investigating the synergy between our work and different software development model to further enlarge the applicability.

Acknowledgment

The research described has been carried out as part of the PHANTOM Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 688146.

References

[1] Capgemini, World Quality Report 2017-2018

[2] "Model-Based Design for Embedded Systems," *Taylor & Francis*. [Online]. Available: https://www.taylorfrancis.com/books/e/9781420067859. [Accessed: 26-Oct-2018].

[3] A. Kramer and B. Legeard, Model-based testing essentials: guide to the ISTQB certified model-based tester foundation level. Hoboken, New Jersey: John Wiley & Sons Inc, 2016.

[4] T. A. Henzinger and J. Sifakis, “The Embedded Systems Design Challenge,” in *Proceedings of the 14th International Conference on Formal Methods*, Berlin, Heidelberg, 2006, pp. 1–15.

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

[6] W. Li, F. Le Gall, and N. Spaseski, “A Survey on Model-Based Testing Tools for Test Case Generation,” in *Tools and Methods of Program Analysis*, 2018, pp. 77–89.

[7] S. Mohacsi, M. Felderer, and A. Beer, “Estimating the Cost and Benefit of Model-Based Testing: A Decision Support Procedure for the Application of Model-Based Testing in Industry,” in 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, 2015, pp. 382–389.

[8] J. Boberg, “Early Fault Detection with Model-based Testing,” in *Proceedings of the 7th ACM SIGPLAN Workshop on ERLANG*, New York, NY, USA, 2008, pp. 9–20.

[9] PHANTOM, http://www.phantom-project.org/

[10] First design for Cross-layer Programming, Security and Runtime monitoring, http://www.phantom-project.org/results

[11] Moscato, F., Mazzocca, N., Vittorini, V., Di Lorenzo, G., Mosca, P., and Magaldi, M. Workflow pattern analysis in web services orchestration: the BPEL4WS example. Proceedings of the First international conference on High Performance Computing and Communications, Springer-Verlag (2005), 395‑400.

[12] “Eclipse Formal Modeling Project,” [Online]. Available: https://projects.eclipse.org/proposals/eclipse-formal-modeling-project. [Accessed: 26-Oct-2018].

[13] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles and C. Willcock, “An introduction to the Testing and Test Control Notation (TTCN-3),” in *Computer Networks, 42(3):375–403,* , 2003 .

[14] “Eclipse Titan.,” 2018. [Online]. Available: https://projects.eclipse.org/projects/tools.titan . [Accessed: 26-Oct-2018].