Distributed Testing of Multicomponent Systems

Boris Tyutin, Igor Nikiforov, Vsevolod Kotlyarov
Saint Petersburg State Polytechnical University, Saint Petersburg, Russia
b.tyutin@ics2.ecd.spbstu.ru, i.nikiforov@ics2.ecd.spbstu.ru, vpk@ics2.ecd.spbstu.ru

***Abstract* — This paper features an approach that brings together testing of multicomponent systems, formal requirement specifications and automated test suit generation in a one technology.**

I. INTRODUCTION

In software development testing is usually performed at the end of the whole process. This traditional approach leads to the increase in the costs of correction of errors found during the testing or the program product piloting. This, together with the high resource-intensiveness of a testing process itself can lower the test coverage of software product and, therefore, its quality. This problem was partially solved by test-driven software development methods [1]. But the latter have strict limitations (related to the development process and coding politics), which make it unproductive to apply them for medium-size and large industrial projects. The main issue of these methods is the complexity of the adjustment and the scaling of software specifications. It is caused by the requirements modification and clarification during the lifecycle of the software product and it dramatically increases the time necessary for testing.

Thus, the possibility of using test suites related to different abstraction levels and their simultaneous execution gives the possibility to perform testing during all stages of the software product development – from the architectural design abstractions elaboration till product release. Testing automation in this case should significantly reduce the testing process costs.

In telecommunication software development it is a common practice to use a distributed architecture due to the peculiarities of the task domain. For such systems, test procedures have to imitate the whole environment or a part of it in order to be efficient. Also, the possibility of substituting some of the system components with test case entities makes it easier to locate an error or a "bottle neck" within the software. In this case, the testing laboratory contains the following parts:

- system under test (SUT);
- testing controlling system;
- monitoring and test data generation system;
- testing agents.

Distributed allocation and remote interconnection of different parts of the testing system add some important properties to the testing process:

- scalability;
- statistics collection and control of the testing process;
- test suite configuration and test result history storing.

This paper reviews the technology of automated software testing based on automated generation of the test suite from the formal specifications of the system in Use Case Maps notation. Also it briefly outlines the usage of symbolic verification technique (UCM Specification Translator or UST and VRS [6]). Finally, the architecture of distributed testing system (TestCommander) is described, which allows automatic test suite execution and adjustment according to the SUT architecture.

II. SYSTEM REQUIREMENTS FORMALIZATION

In order to be able to have an unambiguous interpretation of the requirements during the creation of a test suite, it is necessary to convert these requirements from an implicit or informal form to a notation, which provides semantic invariance when working with the specifications of the system. At the same time, if there are manual steps in the technology, the notation is required to be understandable by the user. The possibility of description of parallel interactions is also required in selected classes of testing tasks. The Use Case Maps (UCM) notation satisfies all these requirements. It was created by a group of scientists in the University of Ottawa [2] at the end of the twentieth century and is widely used for requirements engineering, system design and creation of test scenarios. UCM is used for behavior specification in telecommunication, distributed systems and other areas where it is necessary to specify an intercommunication of different entities. The notation has been standardized in 2003 by ITU-T [4]. The main elements of UCM notation are team (object or agent under observation), responsibility ( , action under observation), start point () and end point () of scenario, "and fork" () and "and join" ( ) (parallel scenario description), failure point (, exception under observation), timer ( , system timer object) and stub ( , is used for hierarchical diagram composition).

UCM project is a set of connected and structured maps containing a sequence of UCM elements. A set of maps describes the behavior of the system just as it is done in the requirements. The example of UCM map for a large telecommunication project is depicted in the Fig.1. There are three agents, one is for incoming request receiving (STMD), other is for request processing (RRt), and the third one performs exception handling (FRMP). In case of receiving of the incoming request (responsibility HandoffRecognized), the

timer (TStfHotDir) is set. If it expires, a corresponding timeout is fired (responsibility StmdHanUnsucc) and an exception occurs (failure point Failure). It is pretty easy to specify the behavior of a multicomponent system in terms of UCM agents and actions. But the way it is formalized depends on the experience and qualification of the engineer who creates the system specifications.

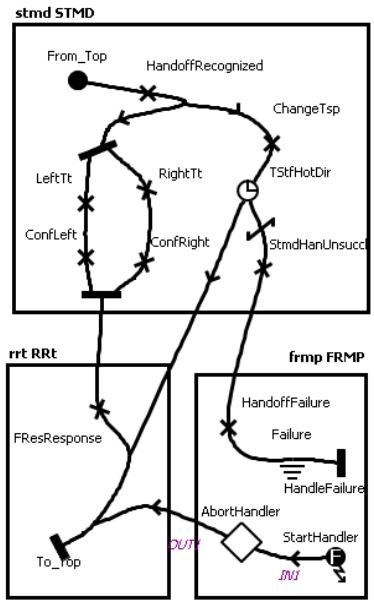

Figure 1. UCM map for a telecommunication project.

UCM notation allows defining system behavior in very convenient graphical way. In order to enable data flow, sending and receiving signal points definition (as well as usage of conditions for alternative behavior and exceptions), UST tool functionality was expanded by metadata grammar. The information from metadata is used during the process of formalization and is fully reflected in the model. Also, the ability to describe environment of the model, where the initial parameters are defined, was added.

For automatic transformation of high level design in UCM notation into formal description in basic protocol language UST [7] tool is used. Its main features are: the UCM analysis of errors, where the syntactic correctness of metadata and UCM structures are checked; the optimization of base protocol system model, which improves the efficiency of the instruments verification and testing; component-based approach of model structuring, which allows verification and test generation to be independent from individual parts of the model.

The tool helps to create the UCM model, add required information to metadata and translate it to base protocols (BP) [6]. One of the most important parts of the tool is UCM analysis module. Developing the UCM model, the user is guided by his understanding of the system requirements, his knowledge and experience in the behavior definition. But despite all the advantages of the developer, the model may have errors associated with user competence in formalization. The most common modeling errors are the ones of incorrect conditions of alternative branches and usage of shared resources by concurrent threads. The analyzer warns the user about potential dangerous places before building a formal model and helps him to make changes in order to remove the warnings.

Completed UCM project is checked for errors by the analyzer. After it is free of errors, it is possible to set up the formalization parameters. After the translation, the BP model is imported into the symbolic verification tool VRS. It performs the verification and the generation of test cases, which cover specified system requirements. Obtained symbolic test scenarios are filled with real data and can be used as a test suite specification for TestCommander tool or visual representation of system behavior.

III. DISTRIBUTED TESTING APPROACH

Full control over a test execution and a test result analysis are required to perform conformance testing of a system. Therefore, it is necessary to create a test suite, which is able to specify the interaction with the SUT in exactly the same way it is done in the requirements. Observing the behavior of the system during the execution of such test cases allows to establish whether the system implementation is correct or not[5].

For this purpose, Message Sequence Chart (MSC) language is used in TestCommander for test suite description [3]. It automatically generates the code of test suite modules on a target programming language (C++, Java, tcl) from MSC charts. These modules interact with SUT and each other using predefined interfaces, reproducing test scenarios. These interactions can be unambiguously and in a human-oriented manned defined in terms of MSC and, thus, represented in textual or graphical view.

The distributed testing method reviewed in this paper relies on automated approach for test suite generation. This approach is based on combined usage of UST and VRS tools and is demonstrated in Fig. 2.

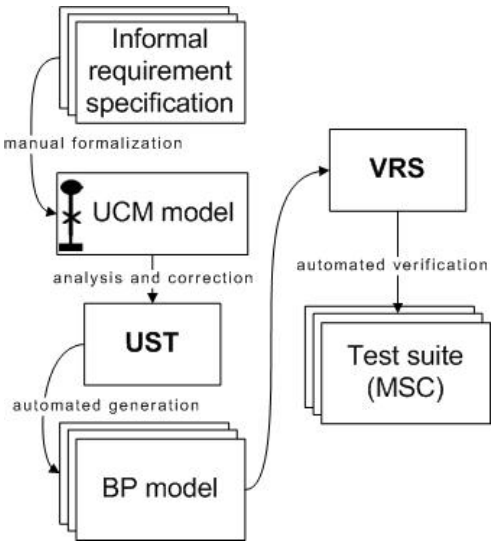

Figure 2. Automated test suite generation tool chain

As a combination of requirements formalization technique described earlier and the technology of test scenario generation based on the verification of a model in the BP notation [9], this approach brings together the requirements management, verification and testing in one technology. Automatically generated tests define the complete description of the interaction of all the components of SUT and its environment. All of the above allows these specifications to be tested and verified.

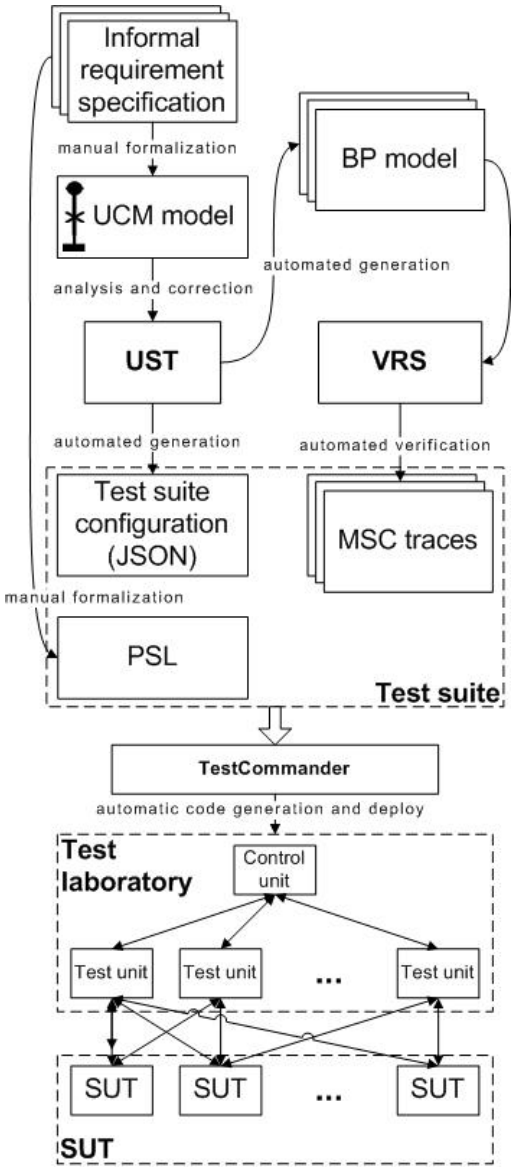

Figure 3. Distributed testing approach overview

TestCommander tool accepts MSC charts obtained with various methods (assuming that they correspond to MSC standard). However, the proposed approach has some significant advantages, such as:

- Automation of routine and time-demanding operations;
- Simple and human-oriented formal notations usage;
- Requirements checking with verification methods [6].

Generated test suite executable file set consists of one or more testing units and one control unit. Testing unit interacts with the SUT according to the test logic and exchanges the control signals with the control unit. The latter controls the test execution process and collects testing results. Protocols of interaction between testing system units and SUT are defined with Protocol Specification Language (PSL). This notation unambiguously specifies the format of the messages passed between the entities involved in the testing. PSL specification is created manually and is used for test suite code generation.

For test suite configuration, code generation and test suite deploy setup a configuration file is used. It is written in JSON [8] and specifies the location of testing units and SUT components. Nonetheless, the main feature of this configuration is to specify, which of the SUT components are substituted with the test units. It allows testing of the behavior only of a part of the system. The configuration file is generated automatically from UCM model, but it can be adjusted manually.

After the test suite is configured, its code in a target language is automatically generated and the test suite is deployed in the test laboratory. Testing starts with the execution of the control module of the test suite, which than controls the test unit threads and SUT. Test results are the MSC diagrams of test activities. The whole process is presented in Fig. 3.

IV. TECHNOLOGY APPLICATION LIMITATIONS

This approach is highly efficient due to the high level of the automation of routine stages of the test suite code creation and the adjustment of the whole test suite in case of any changes in the system specifications. However, as a result of some peculiarities of technologies included in the tool chain, there are limitations of its application in different tasks:

- Formal system specification can be created only in UCM;
- Generation of test cases is performed by VRS verification tool;
- Testing system has distributed architecture.

Pure UCM notation can be used only for functional requirements, which describe the control flow of the program. To alleviate this restriction, new semantic elements were introduced in UST tool – the metadata. It allows working with data flow of the system.

The verification tool VRS uses the state machine representation of the system in base protocol notation. The MSC diagrams are artifacts of the verification process and are used as test scenarios for test suite code generation. For some particular systems, it may take a significant amount of time to cover all the requirements with tests. As a solution for this problem, a new special feature was introduced in UST. By using the UCM model of the system, it can automatically generate special rules used in VRS during the analysis and trace generation. Also, a branch test criteria is used for the test suite optimization.

Another factor is the architecture of the testing system itself. It is based on the remote communication of the test units, the strict format of interconnection protocols and requires the pre-configured testing laboratory (linked workstations with some pre-installed software). Together with the state machine representation-based approach used in the test suite code generation, this allows the approach to be applied with the highest efficiency for the telecommunication system testing.

V. CONCLUSION

Multicomponent software testing problem is the complicated one. Computer-aided software engineering technologies can reduce efforts required for resolving this problem in industrial projects. Automation of different parts of software development process significantly increases the quality of the product. During our work a set of tools (UST, TestCommander) was developed. After integration with the verification system (VRS), it was used to form the semi-automated software testing method. The latter one was applied for various telecommunication projects (such as an implementation of a part of LTE standard).

The method, reviewed in this paper, is a combination of requirement management, verification and testing. It allows performing the checking of the correctness of the system implementation according to its specification within one technology. Testing approach is based on the automatically generated test suite, the correctness of which is proved during the system formal specification verification. It reduces the cost of regression testing needed in case of a changing or a refinement of specifications. All stages of this method are fully or partly automated. The developed software components used in these stages are independent; and all data formats are standardized. All of the above ensures that the whole method is scalable, highly flexible and adaptive and open for modernization. However, the technologies and implementation of the method limit the class of software testing problems, which can be successfully and efficiently solved with the approach reviewed in this paper.

REFERENCES

[1] K. Beck, Test-Driven Development by Example, Saint Petersburg: Piter, 2003.

[2] R.J.A. Buhr and R.S. Casselman, Use Case Maps for Object-Oriented Systems. London: Prentice Hall, 1996.

[3] ITU-T Recommendation Z.120: Message sequence chart (MSC). Geneva, Switzerland, October 1996, http://eu.sabotage.org/www/ITU/Z/Z0120e.pdf.

[4] ITU-T Recommendation Z.151 : User requirements notation (URN) - Language definition. Geneva, Switzerland, September 2003, http://www.itu.int/rec/T-REC-Z.151-200811-I/en.

[5] C. Kaner; J. Falk, H. Q. Nguyen. Testing Computer Software (2nd ed.). New York: Wiley, 1999.

[6] A. Letichevsky et al, “Basic protocols, message sequence charts, and the verification of requirements specifications”, Computer Networks: The International Journal of Computer and Telecommunications Networking, v. 49 n. 5, pp. 661-675, 5 December 2005.

[7] I. V. Nikiforov, A. V. Petrov, Y. V. Yusupov, “Generation of formal model of a system from requirements specified in USE CASE MAPS”, Scientific and technical statements STU. № 4 (103), pp. 191-195, St. Petersburg: St. Petersburg Polytechnic University, 2010.

[8] RFC 4627: The Application/JSON Media Type for JavaScript Object Notation (JSON), July 2006, http://www.ietf.org/rfc/rfc4627.txt

[9] A. O. Veselov, V. P. Kotlyarov, “Testing automation of projects in telecommunication domain”, Proceedings of the 4th Spring/Summer Young Researchers’ Colloquium on Software Engineering, Nizhny Novgorod, June 1-2, 2010, pp. 81-86.