Industrial software verification and testing technology

P.D. Drobintsev

Ph.D. assistant prof., SPbPU
drob2@ics2.ecd.spbstu.ru

V.P. Kotlyarov

Ph.D., prof., SPbPU
vpk@ics2.ecd.spbstu.ru

A.A. Letichevsky

Academician ,Glushkov Instiute of Cybernetic
let@cyfra.net

I.A. Selin

Assistant, SPbPU
selin@ics2.ecd.spbstu.ru

Abstract. This paper is devoted to technology of automating full cycle of software development from natural language processed formalized requirements and their analysis with symbolic verification to tests generation and execution. Suggested technology is capable of checking semantic consistency of requirements in generated code of target application. Primary target areas of technology applicability are telecommunication and distributed systems, but other types of applications can also be developed using the technology.

Keywords: Software quality assurance, formal model, verification, testing automation.

1 Introduction

One of the main problems in the development and test automation of industrial applications' software is handling of complicated and large scale requirements specifications. Documents specifying requirements specifications are generally written in natural language and may contain hundreds and thousands of requirements. Thereby the task of requirements formalization to describe behavioral scenarios used for development of automatic tests or manual test procedures is characterized as a task of large complexity and laboriousness.

Agile methodologies are used more and more often by the software developers. There are different ones: Scrum[1], eXtreme Programming (XP)[2], Adaptive Software Development (ASD)[3] and Crystal[4], but at the core they all have iterative build-up of functionality and releases at the end of every sprint (which usually lasts about several weeks)[5]. Requirements for the system may change during the development, creating difficulties in checking certain release. As said by B. Mayer, main achievement of agile is establishment of tests and mainly regression tests as an essential part of a development process [6].

In this paper, an approach is proposed for creating an effective verification and automated testing technology, which should be used all along the development process. Tests are a part of a project, just as the system code is, and recommended by every agile methodology listed above. Usually the mandatory checks are run on unit tests, which are verifying small parts of the code before integration into main development branch, and functional tests, which are verifying typical scenarios and use cases from the user point-of-view [7]. But this information is insufficient for checking the system behavior, because scenarios and use cases are not taking into account the environment and all possible software settings. There is always a possibility that on some data system will behave incorrectly and it will not be checked by the test suite. A lot of software failures are bonded to special cases which were not covered by tests [6].

This is a common situation, when software developers just do not have enough time or resources to properly test the system in non-standard cases, estimate the dangers of “copy-paste” from one module to another and other types of behavior where checking could take a lot of time and requires a thoughtful code analysis of possible negative outcome.

Because in agile methodologies the development of design specification is omitted, the requirements will take the brunt and thus must be created as good as possible. Thereby the task of requirements formalization to describe behavioral scenarios used for development of automatic tests or manual test procedures is characterized as a task of large complexity and laboriousness.

Applicability of formal methods in the industry is determined to a great extent by how adequate is the formalization language to accepted engineering practice which involves not only code developers and testers but also customers, project managers, marketers and other specialists. It is clear that no logic language is suitable for adequate formalization of requirements which would keep the semantics of the application under development and at the same time would satisfy all concerned people [8].

In modern project documentation the formulation of initial requirements is presented either in constructive way, when checking procedure or scenario of requirement fulfillment can be constructed from the text of this requirement in natural language, or unconstructively, when functionality described in the requirement does not contain any explanation of its checking method.

For example, behavioral requirements of telecommunication applications in case of described scenario of requirements coverage are constructively specified and allow using of verification and testing to check requirements implementation in software. Non-behavioral requirements are usually unconstructively specified and require additional information during formalization which allows reconstructing the checking scenario, i.e. converting of non-constructive format of requirements specification into constructive one.

2 Requirements coverage checking

The procedure of requirement checking is exact sequence of causes and results of some activities (coded with actions, signals, states), which analysis may prove that current requirement is either covered or not. Such checking procedure can be used as a criterion of coverage of specific requirement, i.e. it can become a so-called criteria procedure. In the text below a sequence or "chain" of events will be used for criteria procedure.

Tracking the facts of criteria procedure coverage in system's behavioral scenario (hypothetical, implemented in the model or real system), it can be asserted that the corresponding requirement is satisfied in the system under analysis.

Procedure of requirement checking (chain) is formulated by providing the following information for all chain elements (events):

- conditions (causes), required for activation of some activity;
- the activity itself, which shall be executed under current conditions;
- consequences – observable (measurable) results of activity execution.

Causes and results are described with signals, messages or transactions, commonly used in reactive system's instances communications [9], as well as variables states in form of values or limitations on region of admissible values. Tracking states' changes, produced by chains activities it's possible to observe the coverage of corresponding chains. While analysis it is acceptably to consider a direct transition from a state into a state with a null activity, and in case of non-determinism – alternative variants of states choices and changes.

Problems with unconstructive formulating of requirements are resolved by development of requirement coverage checking procedures on user interfaces or inter-component interfaces.

Thus, chains containing sequences of events can appear as criteria of requirements coverage; in addition, it is possible that criterion of some requirement coverage is specified not with one, but with several chains.

3 Initial documents specifying application requirements

Usually initial requirements in technical documentation are formulated in natural language and can be presented in one of the following form:

- in form of behavioral requirement, scenario (procedure) of requirement checking can be retrieved based on requirement text;
- in form of non behavior requirement, only structure and sense of requirement can be retrieved without information about requirement checking.

Any behavioral requirement can be formalized with further automatic analysis; for this purposes VRS/TAT technology [10] can be used.

In VRS/TAT technology Use Case Maps (UCM) notation [11] (Fig. 1) is used for high-level description of the model, while tools for automation of checking and generation work with model in basic protocols language [12].

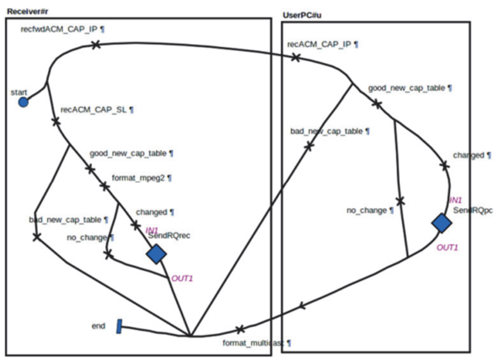

Fig. 1. UCM model of two instances: Receiver and UserPC.

UCM model (Fig.1) contains two interacting instances model description. Each path on the graph from the event "start" to the event "end" represents one behavioral scenario. Each path contains specified number of events (Responsibilities). Events on the diagram are marked with × symbol, while Stub elements which encode inserted diagram – with ♦ symbol. As a result, each scenario contains specified sequence of events. Varieties of possible scenarios are specified with variety of such sequences.

In these terms a chain is defined as subsequence of events which are enough to make a conclusion that the requirement is satisfied. A path on the UCM diagram, containing the sequence of events of some chain, is called trace, covering the corresponding requirement. Based on the trace the tests can be generated which are needed for experimental evidence of the requirement coverage.

4 Traceability matrix

Verification project requirements formalization starts with creation of Traceability matrix (TRM) [8]. "Identifier" and "Requirements" columns contain requirement's identifier, used in the initial document with requirements, and text of the requirement, which shall be formalized. "Traceability" column contains chains of events sufficient for checking of corresponding requirement coverage and "Traces" column with traces or behavioral scenarios used for tests code generation.

4.1 Development integral criteria of requirements coverage

The distinctive feature of VRS/TAT technology – special criteria of each requirement's coverage checking. Below all criteria related to requirements are listed in ascending order of their strength:

1. Events criterion - coverage level in generated scenarios of subset of events used in criteria chains.
2. Chains criterion - coverage level in generated scenarios of subset of chains (consisting of events and states of variables) with at least one for each requirement.
3. Complex criterion - coverage level in generated scenarios of the whole set of chains specifying integral criteria (combined from criteria 1 and 2) of requirements coverage.

Criteria development shall be adaptive to specific project [13, 19, 20]. Criteria shall be applied flexibly and can be changed according to conditions of scenarios generation.

4.2 Generating and selecting scenarios which satisfy to specified coverage criteria

Model-checking based technique is used to generate symbolic and concrete traces (STG and CTG) [14, 15, 16]. To reduce the “explosion” of possible combinations of basic protocols some limitations are introduced in form of Guides [17].

4.3 Guides creation. Automatic and manual processes of guides creation

There are two ways of guides creation: automatic and manual. Pros and cons of both processes are described in [17].

4.4 Technology chain of test scenarios generation

The process of tests generation with usage of VRS/TAT tools can be divided into following phases:

Manual creation of formal model for requirements in UCM language. Key actions are refined from initial requirements. Based on these actions a set of chains, which will be translated into behavioral scenarios are created. On the next step objects which interact in the scope of scenarios and structure are refined. Metadata for data flow are added into formal model. More detailed description of the phase is presented in [13].

The next phase is conversion of UCM model into BP notation [10] which is input language for VRS/TAT tools. Translation is performed with saving of model construction semantics and as result of this phase equivalent to initial UCM model is generated.

Guides which satisfy branch criterion are also generated based on initial UCM model. In process of generation all branches are traversed with VRS/TAT toolset.

On the next phase formal model in basic protocols notation and guides are used in trace generator of VRS tool for verification and automatic traversal of behavioral tree for symbolic scenarios generation.

After scenarios generation process finished it is necessary to analyze coverage of guides by traces with goal to define which guides are still not covered and identify the reasons. Such analysis can be performed in EVA module. Based on analysis results UCM model or guides shall be corrected.

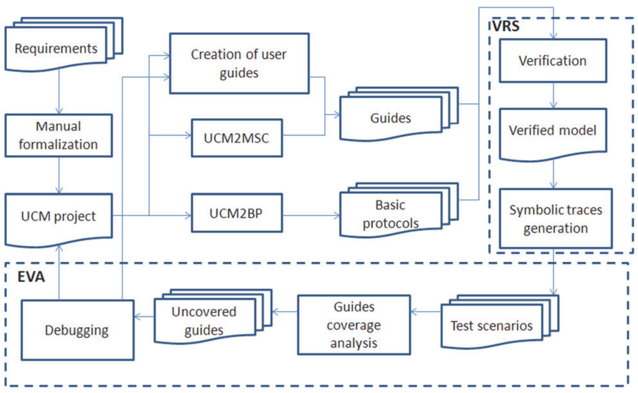

Fig. 2. Process of test scenarios generation with usage of static analysis tool EVA.

5 HPC deployment

The amount of generated test scenarios may be very big and thus running the whole toolchain may require a lot of time, which is not desirable and can significantly slow down all development process.

The most obvious way to speed the things up is to run tools in parallel. To do this, each of the processing steps was divided on separate parts:

- Parallel traces generation became possible thanks to guides [18, 21]. Guides are independent, so after basic protocols model is obtained, there are as many VRS instances being executed, as there are number of guides.
- Parallelization of concretization step is made by executing script on each symbolic trace in parallel
- Lowering step parallelization made in the same manner as in concretization step. Concrete traces are processed in different threads, as they are independent [21]
- Running test suite was made by launching many instances of TAT and SUT. Running each test requires launching SUT and TAT. After test is done, SUT must be restarted in order to reset its state to initial.

Theoretically, independent process parallel launching will give linear scalability. However, there are still limits on executing on CPU/server, because machine has limited number of cores/threads. Therefore, you will only have some work running, while the rest of it will wait for its turn. Of course, you can improve the performance of the system by adding another set of computing nodes. By bringing more and more computing power, we are getting closer to the term of high performance computing (HPC).

The use of supercomputer can help to extend the number of testing instances to the point where we can run all tests from suite simultaneously (or at least a lot of them).

Test suite was divided into several parts (by the number of nodes), which were processed individually on each node, which includes launching testing toolset and running the SUT.

6 Results

Project contained stats: 2500 basic protocols, 42000 symbolic traces, 160000 concrete traces and tests was released by S.Petersburg Polytechnic University's Supercomputer [22]. Supercomputer allow resources for running 10^3 – 10^4 tests in parallel. Test suite was divided into several parts, which were processed individually on each node. Each node was running its own test job, which includes launching testing toolset and running the SUT. Result - we can run all tests from suite simultaneously (or at least a lot of them).

7 Conclusions

The result of this work is improved technology which integrates verification and testing of software projects and provides:

- Full automation of industrial software product development process with the control of requirements semantics realization.
- Generation of application's model and symbolic behavioral scenarios, which fully (100%) cover behavioral features of the application.
- Automated concretization of symbolic traces in accordance with test plan.
- Automated execution of system and regression testing phases.
- Supercomputer resolves problem of testing performance and we can run all tests from suite simultaneously
- High level of automation for the process of development and managing of software product quality.

Results of integrated design and testing technology appliance in the development of wireless telecommunication applications demonstrated 26% time-saving in software product creation.

Acknowledgments. The work was financially supported by the Ministry of Education and Science of the Russian Federation in the framework of the Federal Targeted Programme for Research and Development in Priority Areas of Advancement of the Russian Scientific and Technological Complex for 2014-2020 (№ 14.584.21.0022, ID RFMEFI58417X0022).

References

1. K. Schwaber, M. Beedle. Agile Software Development with Scrum – Prentice Hall – 2002.
2. K. Beck. Extreme Programming Explained: Embrace Change – Addison-Wesley Professional, 2000. - 194 p.
3. J. Highsmith. Adaptive Software Development: a Collaborative Approach to Managing Complex Systems - Addison-Wesley – 2013
4. Cockburn A. Crystal Clear: a Human-Powered Methodology for Small Teams –Pearson Education – 2004 – 312 p.
5. Principles behind the Agile Manifesto. URL: http://agilemanifesto.org/iso/en/principles.html
6. B. Meyer. Agile!: The Good, the Hype and the Ugly – Springer – 2014 – 170 p.
7. J. Hanley. Scrum - User Stories: How to Leverage User Stories For Better Requirements Definition (Scrum Series) (Volume 2). – CreateSpace Independent Publishing Platform – 2015 – 116 p.
8. S.Baranov, V.Kotlyarov, A.Letichevsky. Industrial technology of mobile devices testing automation based on verified behavioral models of requirements project specifications// «Space, astronomy and programming» – SPbSU, SPb. – 2008. – pp. 134–145. (in Russian).
9. Z.Manna, A.Pnueli.: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1992.
10. S.Baranov, V.Kotlyarov, A.Letichevsky, P.Drobintsev. The technology of Automation Verification and Testing in Industrial Projects. / Proc. of St.Petersburg IEEE Chapter, International Conference, May 18-21, St.Petersburg, Russia, 2005 – pp. 81-86
11. Recommendation ITU-T Z.151. User requirements notation (URN), 11/2008
12. A. Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V. Kotlyarov, T. Weigert. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. Proc of ISSRE04 Workshop on Integrated-reliability with Telecommunications and UML Languages (ISSRE04:WITUL), 02 Nov 2004: IRISA Rennes France.
13. P.Drobintsev, V.Kotlyarov, I.Chernorutsky. Test automation based on user scenarios coverage. “Scientific and technical sheets”, SpbSTU, vol.4(152)-2012, pp.123-126 (in Russian).
14. I.Anureev, S.Baranov, D.Beloglazov, E.Bodin, P. Drobintsev, A.Kolchin,, V.Kotlyarov, A. Letichevsky, A. Letichevsky Jr., V.Nepomniashiy, I.Nikiforov, S.Potienko, L.Priyma, B.Tytin. Tools for support of integrated technology for analysis and verification of specifications telecom applications // SPIIRAN proceedings- 2013-№1-28P.
15. A.Kolchin, V.Kotlyarov, P. Drobintsev. A method of the test scenario generation in the insertion modelling environment // “Control systems and computers”, Kiev: "Akademperiodika", vol.6-2012, pp.43-48 (in Russian)
16. A.A. Letichevsky, J.V. Kapitonova , V.P. Kotlyarov, A.A. Letichevsky Jr., N.S.Nikitchenko, V.A. Volkov, and T.Weigert. Insertion modeling in distributed system design // Programming problems. – 2008. – pp. 13–38
17. V.P. Kotlyarov, P.D. Drobintsev, I.V. Nikiforov. Integrated technology for industrial software verification and testing // Proceedings of the 2014 International Conference on Mathematical Models and Methods in Applied Sciences (MMAS ’14) – Saint Petersburg State Polytechnic University, Saint Petersburg, Russia – September 23-25, 2014 – pp.138-145
18. A. Letichevsky Jr., A. Kolchin. Test scenarios generation based on formal model // Programming problems. – 2010. – № 2–3. – pp. 209–215 (in Russian)

19. V.P. Kotlyarov. Criteria of requirements coverage in test scenarios, generated from applications behavioral models // “Scientific and technical sheets”, SpbSTU. – 2011. – vol.6.1(138). – pp.202–207. (in Russian)
20. Baranov S., Kotlyarov V., Weigert T. Varifiable Coverage Criteria For Automated Tesdting. SDL2011: Integrating System and Software Modeling // LNCS. –2012– Vol.7083 – P.79–89.
21. P.Drobintsev, V. Kotlyarov, I.Nikiforov, N.Voinov, I.Selin.: Conversion of abstract behavioral scenarios into scenarios applicable for testing. SYRCoSE-2016, ISPRAS, pp. 96-101
22. Creating "Polytechnic RSC Tornado" supercomputer for St. Petersburg State Polytechnical University. URL: http://www.rscgroup.ru/ru/our-projects/240-sozdanie-superkompyutera-politehnik-rsk-tornado-dlya-spbpu