Testing Object-Oriented Systems by Using a Random Sequence of UML Diagrams

Anna Mroczek

Cracow University of Technology; Krakow, Poland

Summary
UML (Unified Modeling Language) as a standard as a standard of specification of object systems should be a natural source of information relevant to the testing. However, the systems are usually very complex. Moreover, the UML models contain a lot of data which are difficult for formalization and require human assistance. This makes ATPG (Automatic Test Pattern Generatio) from UML very doubtful. Consequently, random testing might be an interesting alternative for ATPG.

Key words:
UML, random testing, object-oriented system, test scenarios.

1. Introduction

OOS (object-oriented systems) facilitate the change in the system in order to improve the functionality of the system. This allows the re-use of code which was previously applied in each of the subsystems. It also facilitates the integration of subsystems to a large system, as well as the design of distributed systems [1].
UML was adopted by the Object Management Group (OMG) and it is a standard for modeling the object-oriented system (OOS). Unified Modeling Language is used to specify, visualize, modify, construct, and to document the artifacts of the object-oriented software-intensive system during its development. Unified Modeling Language (UML) combines the techniques of modeling entity–relationship model (ER model), activities (Data flow diagram - DFD), Object -Oriented Analysis (OOA) and management complexity. UML allows you to present a plan of the system, both the system functions as well as the detailed information about the system [3].
Each system must meet all of the functions set by the user. Therefore, during the development of the system, it should be thoroughly verified and tested. One should check the correctness of the specification to validate the model object net model (ONM) [7].
Another way is to use the evaluation of test scenarios. When the user meets the requirements of the specification, it can be concluded that the system is correct. However, the development of scenarios will be compared to the specification [9].
Each instance of the class is tested at the individual level. They are used in UML diagrams, object states and transitions between states. The method presented in the work of Abdurazik and Offutt [10] shows that the relationship as an external event affects the behavior of the object.
The rationale OOSs at work [11] is a method of a semi-automatic generation of test paths. This method uses the objects appearing in the OOS and their interactions. Because objects perform all the operations of the system, it can be interpreted as a sub with its own attributes and operations. In [11] ONM was presented, as well as the way in which the test paths that represent the behavior of objects can be generated.
In my publication I present a method of random testing of object-oriented systems which is based on their UML models: use case diagrams, sequence diagrams, and class diagrams.
The paper is organized as follows. In section 2 states the problem. Section 3 we briefly discuss related work. Section 4 gives an overview of the method and sections 5 describes the main steps of the random testing. The paper ends with conclusions.

2. Problem statement

Structural testing requires a good definition of the error codes and it is connected with the necessity of defining test vectors for complex errors. This process is very time-consuming. There is also a possibility of omitting the unknown error codes. Functional testing also possesses some limitations on its complexity. Hence the popularity of random testing (pseudorandom testing). It is based on generating an arbitrary subset of input vectors. It can be reasoned that the longer testing sequence is generated, the better error coverage can be obtained. What is more, we do not have to be familiar with the error codes. This simple idea can be proven in practice. The most important thing here is to select an appropriate length of such a sequence in order to obtain the required error coverage.

Manuscript received November 5, 2013
Manuscript revised November 20, 2013

3. Testing OOS

OOS is the subject of many studies. First of all, it is used in the context of UML models as a source of test cases. To a large extent, these studies implement the deployment and testing of integrated systems.

Testing of the system involves the testing of the whole system according to its specifications. It comprises many activities, such as functional testing (the study with a preservation of system specifications) and testing the system efficiency (time to respond as well as the use of resources), [5] In other words, the way the test is performed is compared with the aim of the specification.

In the object-oriented context, the UML artifacts take the advantage of the development of UML, so the artifacts (use case, sequence, interaction diagram, class diagram, and object diagram) may be used to analyze operations on the system. Therefore, the most suitable testing scenarios can be obtained [9]. For example, use cases, their relevant sequence or interaction diagrams and class diagrams can be used as a source of valuable information for testing purposes. UML diagrams are a potential source of information in order to generate testing scenarios or to test OOSs. It can be proven by many scientific publications. The publications relate to unit testing as well as to implemented and integrated systems.

OSS testing can distinguish six levels at which software testing occurs – unit test, module test, integration test, subsystem test, system test, and acceptance test [7].

The first level (a unit test) evaluates the validity of the way in which particular units of the system work. In this case a state diagram can be used. It displays particular states of the object, how they are changed, and conditions under which there occur, as well as the hierarchy of states.

The paper presents techniques of Abdurazik and Offutt that use Outt's state-based specification test data generation model to generate test cases from UML state charts. The above mention model involves the conversion of state diagrams into transition tables. [3] It also proposes testing criteria for generating test cases according to state diagrams.

Kim and Hong proposed the application of state diagrams in UML to class testing. A set of coverage criteria is proposed based on control and data flow in UML state diagrams and it is shown how to generate test cases satisfying these criteria from UML state diagrams [8].

The requirements of analytical models, such as a use case model, an interaction diagram, or a class diagram, may be used to generate tests. The recent publications suggest the use of the sequence of messages between objects and UML diagram sequences, so that they may be joined by a category zone of the study [9]. UML sequence diagrams may be also used to generate tests and to study the object integration. The diagrams are used to generate test cases in order to identify the most erroneous way of testing the software. Shanthi proposed the use of SDT and a genetic algorithm in order to obtain the aims of diagram sequence studies [10]. On the other hand, Sokenou [11] proposed the method which applies state diagrams and sequence diagrams at the same time. In his work, Sokenou presented a sequence diagram as a collection of tests, where state diagrams are used in order to add information about participating objects. The tests which are generated due to this method may be used in class as well as integration testing. Testing of the level of integration consists in interface testing and the integration between modules and systems. UML is used here by means of interaction diagrams (sequence or collaboration). Abdurazik and Offutt research [12] used testing in accordance with a traditional control and analysis of the flow of data in collaboration diagrams.

System level testing is performed in order to check whether once an integrated system fulfills as a whole the requirements of the specification. The system is tested as a whole by the use of black box testing. The knowledge about the code and the internal structure of an application is not required at the system level. Testing of the system is the first level at which the system can be tested as a whole. At the lower levels (unit tests, integration tests), particular components and interfaces between them can be tested.

Many publications proposed methods of system level testing which were based on different UML diagrams [13, 14, 15]. The methods mentioned above were differences in the use of UML diagrams: activity diagrams [15] or a Flow Graph [14], which were used as the base artifacts for testing. Sarma and Mall [16] also presented the method which used use case diagrams and sequence diagrams. A UML use case diagram is transformed into a graph called a use case diagram graph (UDG), while a sequence diagram is transformed into a graph called a sequence diagram graph (SDG). Then UDG and SDG are integrated in order to form the System Testing Graph (STG). STG is used with the purpose of generating use cases which may indicate the relation between case, interaction, and defect scenarios.

In my publication uses UML models: use case diagram, sequence diagrams and class diagrams. A use case diagram shows a set of use cases and actors (a special kind of class) and their relationships. An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that are sent among them. Interaction diagrams address the dynamic view of a system. A sequence diagram emphasizes the time ordering of messages.

4. Estimating random test sequence length using UML diagrams

The main problem when testing the system with random sequences, the problem is the reliability of the testing and estimation of a random sequence length required to achieve the desired level of quality tests.

Quality testing (Q_T) can determine the quality of the tests that are written for the test application. Quality tests can measure two parameters: coverage code (code coverage) and coverage of cases (case coverage). This method (cover cases) during the test design is created and describes the test cases and test data. Test case contains set of input values, initial conditions, expected results and created the conditions for the end to cover the objectives of the tests or test conditions. "Standard for Software Test Documentation" [IEEE STD 829-1998] describes the contents of the test design specification and the specification of test cases.

The quality of testing is also being exploited in the testing method random [19]. In this method, test scenarios considered in a system in which errors can occur r set by = {b1, b2, ..., bar}. They assumed that they were tested with random sequence for lengths L. All possible random sequences of length L are included in the set S consisting of $|S|=2^{nL}$ components, wherein S is a subset S_k, b_k error detection.

The probability p_k of this that the random sequence does not test the error b_k is equal [20]:

$$p_k = 1 - \frac{|S_k|}{|S|} \tag{1}$$

The probability p of this that the random sequence does not test all errors fulfills the inequality:

$$\max_k p_k \le p \le \sum_k p_k \tag{2}$$

Testing the system with the random sequence one receives the result affirmative or negative. He qualifies the state testifying about the correctness or the error of the system. Specifying or the system is positive or is negative an inaccurate notion. Because also one ought to pay attention with which probability one can infer about the correctness of the system on the ground the affirmative result of the test or with which probability one can ascertain that the system is incorrect if the test fell out negatively. Thereby one introduced the measure of the quality of testing (Q_T). A measure of the quality testing [19] called is the probability of this that from testing one receives the correct result only then, when tested system will be efficient. Proved that $1\text{-}Q_T$ was equivalent 1-p, and the value Q_T results by definition of probabilities p and p_k. In order to provide the positive result of testing for the credibility one should be sure that the random sequence is testing all mistakes being included in a harvest B. In compliance with [19] the uncertainty for testing system equals p and determine the length of the random sequence. To estimate of the required length of the random sequence uses the information on the probability of detectable error detection hardest.

The probability p_k (probability of detectable error detection hardest) of detecting k error is distinguished as a conditional probability, where a generated vector x belongs to T_k a test vector set of k error. In this case equal:

$$p_k = \frac{1}{n} \cdot \frac{1}{k} \tag{3}$$

where n -the quantity to enter, k - the quantity of errors.

For such testing is fatal the error whose the detection is less probable. From here the uncertainty is equal $\max_k p_k$.

The quality of the detection Q is equal to the probability, wherein the inefficient system was identified as incorrect. In practice belongs so to assure the proper good quality of the detection of errors (4):

$$Q_D = \left|1 - \max p_k\right| = \max \frac{\left|\bar{S}_k\right|}{|S|} \tag{4}$$

where $|S|=2^{nL}$ is constants for all errors,

from here the quality of the detection Q size depends on $|\bar{S}_k|$. In order to simplify determine the length of the random sequence adopted a constant Q value equal 10^{-3}.

Probability random of the test here is equal to p if the error is detected by b_k scenario and 1-p the probability of not random the test. However the probability of not random by of this test of this test sequence L is equal to be equal to $(1\text{-}p)^L$, what can be recorded in the form:

$$Q_D = \max_k p_k = (1-p)^L \tag{5}$$

Transforming the dependence (5) can be estimated the length of the random necessary sequence for the assurance of required quality of testing, which is:

$$L = \frac{\log Q_D}{\log(1 - P_k)} \tag{6}$$

where P_k is a probability of the detection of the most difficult detectable error and is equal p.

This type of testing never gives the results (the validation of a given result) which are at 100% correct. It is always less than 100%. The above mentioned type of testing is bound to improve the system performance.

The given method is designed to estimate and to test the undetectable error in the object system by the use of UML diagrams. It should be proven which of the tested paths

give greater and which of them give lesser probability to be chosen. To that end, tests will examine random critical paths.

The method suggested above consists of two stages:

- Making a table with the data from diagrams – concerning use cases, sequences, classes – and assigning them to particular chart records.
- Making a pseudorandom test and checking its accuracy, estimating the probability and the length of the sequences.

By means of UML diagrams (use case diagram, sequence diagram, class diagram), the user creates a file with data concerning actors of the system, use cases, text scenarios and objects/methods. The information is stored by means of tables which are marked with a unique label. Below we have an algorithm for this operation:

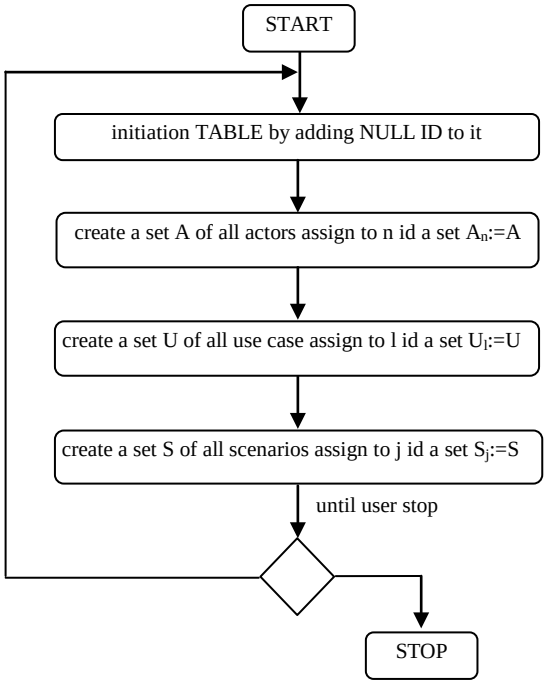

Fig1. Algorithm operation

The following method enables pseudorandom testing of the object system by the use of data from UML diagrams. The program loads a file with data and verifies its accuracy. Then the user makes a choice of particular elements to be tested. First the method of testing should be selected. It may be done in a way- by testing with text scenarios. After the selection of the method of testing, the main part of the program can be developed. Among all actors of the system, one of them should be selected. After that, a use case is assigned to a selected actor. Only then a text scenario can be selected. When all of the elements are approved, there is the testing of scenario. When the test is over, the information about its results, probability, and length of random path is given. The user may cancel testing at any stage of the process. Below presents a method for testing to figure 2.

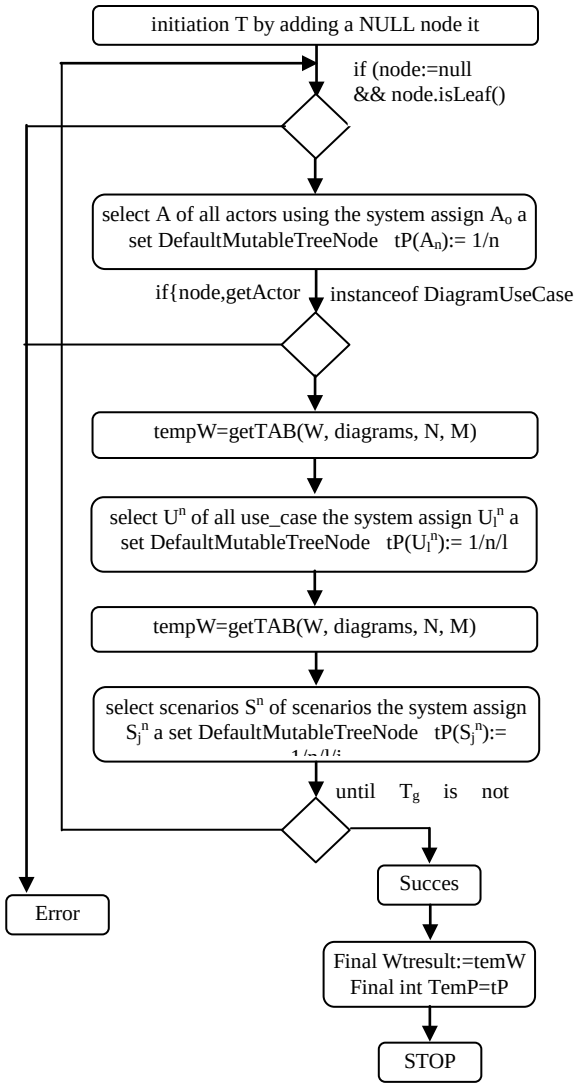

Fig 2. Algorithm operation

The discussed method tests the system in the pseudorandom manner. The probability (P) of the every tested object is calculated and the length (L) of random path is established. The undetectable error from the example is located in the testing path.

5. Example

This chapter Provides an example of a system of object-oriented ATM. An example will be used for analysis and random testing of the system. Used UML diagrams: use

case diagrams, sequence diagram and class diagram. Also used in test scenarios.

The application examines the random testing of object-oriented systems based on their UML models (use case diagram, sequence diagram and class diagram.)

The configuration file must contain the following syntax as in the example configuration file, as follows: ID none of the elements cannot be repeated. If any id occurs twice this error message appears. In this file, you must specify how the system testing.

Create a file with data in UML diagrams. In the figure 3 shows a diagram of a use case. Distinguished three actors: Customer (A_1), Operator (A_2), and Bank (A_3). Each actor has an identifier A_i, where i is the number of the next actor.

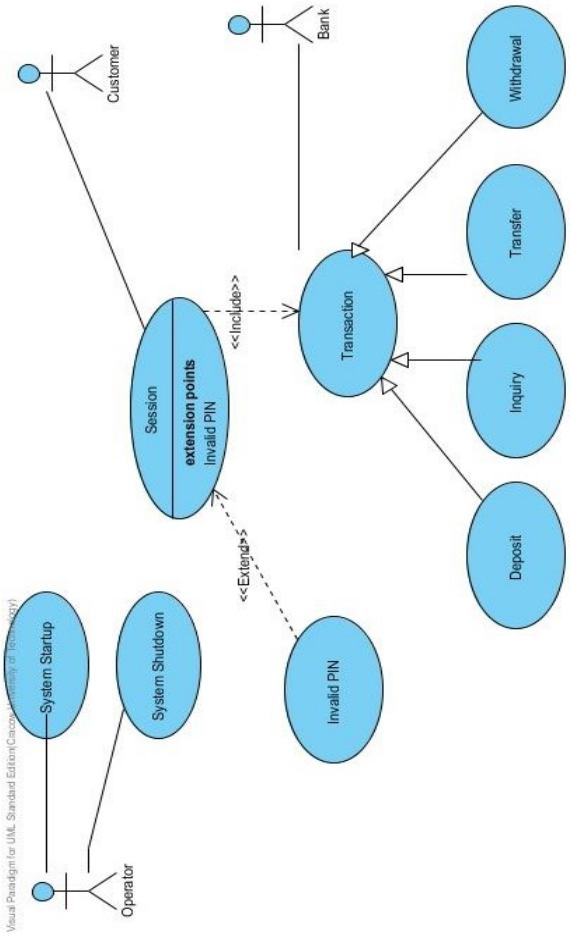

Fig 3. ATM use case diagram

Next step selects random actor A_1 Customer. The use case diagram created table 'Table Uses case'. Each use case has an identifier U^i_j, where j is a sequence number, j is the actor associated with this use case. Then select U^1_1 the use case for the actor - Session. On the Figure 4 (Session Sequence Diagram) shows the sequence diagram for the use case.

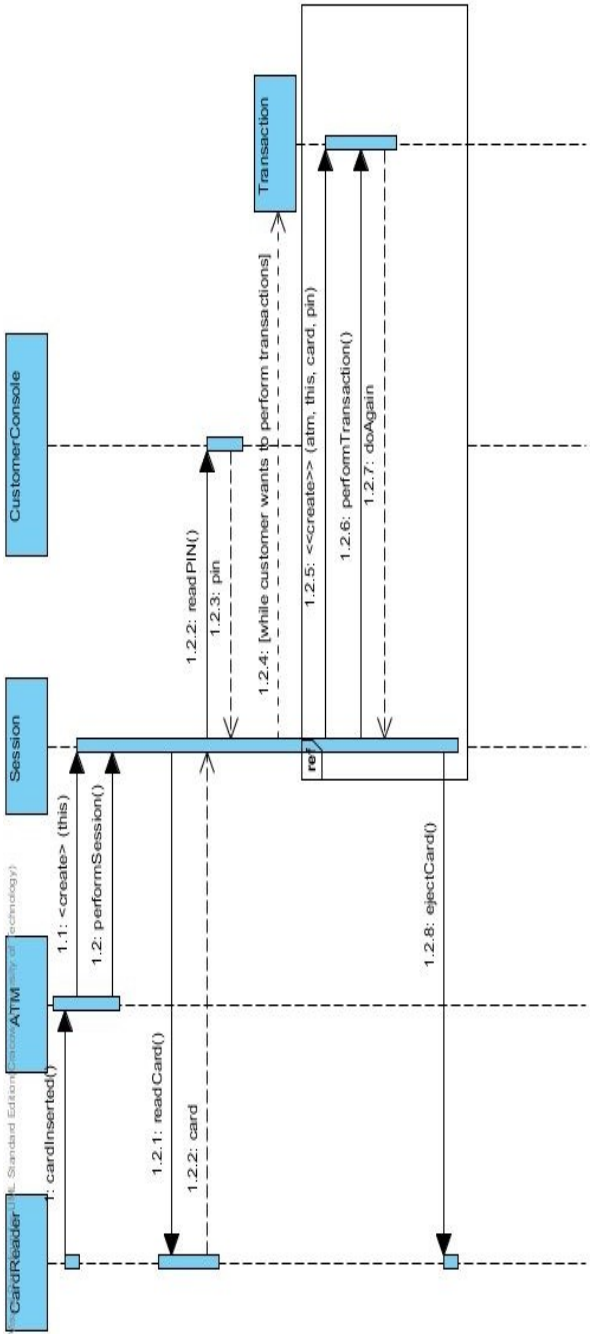

Fig 4. Session Sequence Diagram

Then assigned identifiers S_n^j test scenario, where n is another scenario number and j is the ID of the use case. The next step: selection of the test scenario S_1^1 (Invalid PIN). The table shows the test scenarios for ATM. Following the initial test cases can be Identified early in the design process as a vehicle for checking the Implementation That is basically correct.

Table 1: Margin specifications

Use Case	*Function Being Tested*	*Initial System State*	*Input*	*Expected Output*
System Startup	System is started when the switch is turned	System is off	Activate the "on" switch	System requests initial cash amount
System Startup	System accepts initial cash amount	System is requestin g cash amount	Enter a legitimate amount	System is on
System Startup	Connectio n to the bank is established	System has just been turned on	Perform a legitimate inquiry transactio n	System output should demonstrat e that a connection has been established to the Bank
System Shutdow n	System is shut down when the switch is turned "off"	System is on and not servicing a customer	Activate the "off" switch	System is off
Invalid PIN Extension	Customer is asked to reenter PIN		Enter an incorrect PIN; Attempt an inquiry transactio n on the customer's checking	Customer is asked to re-enter PIN

The next step: selection of the test scenario S_1^1. Select Transaction scenario. Testing scenario. Available all the actors of the use case diagram. Choice actor A_2 (Operator). Then the choice of case use U_2^2 (System Shoutdown). Then select the next item. Follow the guidelines. The algorithm terminates when it reaches the final outcome.

After approval of the summary. The results of probability (from the equation 3) and length (from the equation 6) of the random sequence. Information concerning the probability of drawing a path and the length of random sequences. Probability values of detectable error detection hardest are P_k=1/6 (from the equation 3) (P_k= 0,1667), random sequence of length L= 37,888 (from the equation 6). While, random sequence of length is L = 369, 554 and it is the longest path in the example ATM.

6. Conclusions

Testing of systems demands applications on the examined system of the sequence appropriate testing. Test vectors can be generated for the definite gathering of errors defined on the level of the logical structure (test structure) or so that to check functions realized by the system (testing functional). Both these approaches have their own advantages and defects, both on the level of the generation of vectors of tests, as and costs of tests. Because of this work the attention was sacrificed to testing of random whose an advantage is simple generators and the possibility of composite detections. Very essential is here however the quality of the length of the test which warrants the suitable fault coverage. The smaller the probability of detectable error detection hardest the higher the quality of the test.

Introduced method of testing of random permissible to test the system into the simple and quick manner, obtaining good results of testing.

The discussed method tests the system in the pseudorandom manner. The probability (P_k) of the every tested object is calculated and the length (L) of random path is established. The undetectable error from the example is located in the testing path.

The progress of the technology courses that the sphere of testing and the reliability dynamically is developed because following works they will develop this method.

References

[1] Booch G., Maksimchuk R., Engle M., Young B., Conallen J., Houston K., Object Oriented Analysis and Design with Applications, Pearson Education 2007

[2] Dockerill K., The Importance of Animation with UML, 9th International Symposium of the INCOSE, Brighton, 1999

[3] Myers G.J., Sandler C., Badgett T., Thomas T.M., The art of testing, Wiley, 2004

[4] Abdurazik A., Offutt J., Generating test cases from UML specifications, In: Proc. 2nd International Conference on the Unified Modeling Language (UML99) , Fort Collins, CO, 1999

[5] Sapiecha K., Strug J., Automatic Test Paths Generation from UML Models, 4th IFIP TC2 Central and East European Conference of Software Engineering Techniques, Cracow, 115-128, 2009

[6] Binder R. V. , Testing Object-Oriented Systems - Models, Patterns, and Tools, Addison-Wesley, 2000

[7] Addison-Wesley, 1999. Carleton University TR SCE-01-01-Version 4 Revised, 2002

[8] Briand L., Labichen Y., A UML-Based Approach to System Testing, 2002

[9] Beizer B., Software Testing Techniques, Van Nostrand Reinhold, New York, 2nd Ed., 1990

[10] Kim Y.G., Hong H.S., Cho S.M., Bae D.H., Cha S.D., Test Cases Generation from ULM State Diagrams, IEEE Proceedings -Software 146(4), 187-192, 1999

[11] Ostrand T. J., Balcer M. J., "The Category-Partition Method for Specifying and Generating Functional Test," Communications of the ACM, vol. 31 (6), 676-686, 1988

[12] Shanthi, Kumar , Automated Test Cases Generation from UML Sequence Diagram, 2012 International Conference on Software and Computer Applications (ICSCA 2012), 2012

[13] Sokenou D., Gnerating Test Sequences from UML sequence Diagrams and State Diagrams
[14] Offutt A. J., Abdurazik A., "Using UML Collaboration Diagrams for Static Checking and Test Generation," Proc. 3rd International Conference on the Unified Modeling Language (UML'00), 383-395, October, 2000
[15] Booch G., Rumbaugh J., Jacobson I., The Unified Modeling Language User Guide, Addison Wesley, 1999
[16] Salem A.M., Balasubramaniam L., Utilizing UML use case for testing requirements, International Conference on Software Engineeing resarch and practice, Las Vegas, USA, 2004
[17] Meyer B., "Design by Contracts," IEEE Computer, vol. 25 (10), 40-52, 1992
[18] Sarma M., Mall R., Automatic Test Case Generation from UML Models, 10th International Conference on Information Technology, Roulkela, India, 2007
[19] Dictionary of terms associated with testing, version 2.0, ISTQB, 2006
[20] Sapiecha K., Testing and diagnosis of digital systems, PWN, 1987

Anna Mroczek. Research and teaching assistant of Laboratory of Computing Science, Faculty of Electrical and Computer Engineering, Cracow University of Technology. She received the M. Sc. degree in Applied Computer Science from AGH University of Science and Technology in Cracow, Poland, in 2010. Her research interest includes diagrams UML, modeling and testing object of systems.