# Development of a methodology for cost optimization of software testing for the automatically tests generation

Alexander A. Platonov[1] and Natalia A. Mamedova[1]

[1]*Plekhanov Russian University of Economics, Stremyanny lane, 36, Moscow, 117997, Russia*

**Abstract**

The paper is devoted to issues of software development and support processes, in particular, for software testing. The task of developing a methodology for testing has been solved. It allows optimizing the company's costs for the automatically tests generation. The widely applied software quality control practices were analyzed; the requirements for the developed method were formulated. The IT infrastructure for software quality control has been designed on the basis of the developed methodology. A new result of the research is tools representation for the automatically qualitative tests generation that does not require a significant transformation of the architecture of the software under the test. The presented solution can help companies to optimize significantly their costs for quality control of the developed software. The object for the analysis and implementation of solutions was a company that is a specialized depository, i.e, a participant of the securities market. Its information system is a complex multi-module product. The best practice for increasing the reliability of software, i.e., code coverage with tests has been implemented for it. The task was to find automatically generating tests solution to support the business processes of a specialized depository.

**Keywords**

Automatically generating tests, software, IT architecture, code coverage by tests, business processes, specialized depository



## 1. Introduction

In accordance with ISO/IEC 12207:1995 Information technology – Software life cycle processes, testing is an important part (stage) of the software lifecycle (hereinafter referred to as software). It helps to control the stability of its work and prevent incorrect edits to its source code. Testing is an evaluation of the correspondence between the actual behavior of a program and its expected behavior in the specific, artificial conditions [1]. A stage of software testing is decisive for the company chosen as an object of research. This company is a specialized depository, i.e., a participant in the securities market, which carries out depository activities. Such a company also performs functions of legal support when working with investment funds, mutual investment funds, and non-state pension funds [2]. The range of capabilities (functions) of the information system (hereinafter refers as IS) of the company is very wide. The total number of actions available for the realization is constantly changing. The number of functions was about seven hundred for the period of the study.

The largest and most important components of the company's IS are maintaining a complex of immutable and changeable reference books; generating reports; import, export and synchronization of data; performing actions on a schedule; electronic voting; calculation of the cost of transactions carried out per day; maintaining a set of documents with strictly defined rules for adding, changing and deleting data.

III International Workshop on Modeling, Information Processing and Computing (MIP: Computing-2021), May 28, 2021, Krasnoyarsk, Russia

EMAIL: anonimax@bk.ru (Alexander A. Platonov); nmamedova@bk.ru (Natalia. A. Mamedova)
ORCID: 0000-0002-7780-7019 (Alexander A. Platonov); 0000-0002-8934-7363 (Natalia. A. Mamedova)

Taking into account the scale of IS and the high cost of getting into the production version of undetected errors, it becomes obvious that testing organization is a sophisticated, complex process that takes place every day and requires close attention. The testing database of the company's IS contains several thousand tests and it continues to grow. Maintaining the relevance and completeness of this database began to take up a significant share of the developers' time; so the company needed to automate this process.

Any companies where IS is a complex multi-module product that provides the implementation of business processes face similar problems. Such systems are in constant development and require the support of the required software quality. The introduction of tools to ensure the stability and reliability of the IS today helps to minimize losses in the event of incorrect operation of one of the modules. Modern systems are designed in such a way that adding a new action does not require any additional developer manipulation, except for writing the code of the action itself. Therefore, the incorrect operation of one of modules will not stop the work of the entire IS of the company. However, companies tend to avoid a minimum risk.

Accordingly, each change in the code must go through the stage of automatic testing before becoming part of the deployed system. In the context of IS quality control, the most interesting and effective is regression integration testing of data processing algorithms [3, 4] applying software tools and frameworks working with C # and CLR. For the period of the study, a set of the applied tests included about two thousand units and integration tests. Developers create them, and it takes up to 20% of their working time. The company's management decided to automate the process of developing and maintaining automatically generating tests. This is the optimal strategy in the life cycle of a company 's IS, developed in accordance with the principles of Continuous Integration [5].

One of the well-established practices for improving software reliability is code coverage by tests. Automating the process of tests writing is a separate task that involves the automatically generating test code. Of course, the tests development by a professional tester already has automated alternatives [6]. Attempts to develop universal test generation tools can be found both in public open source projects and in finished commercial products [7-9].

The prospect of applying such products as ProgramExploration, ParasoftdotTEST, NStub was evaluated. These products help to create tests applying a white box method [10], i.e., they are based on the existing code. It can be useful for creating a base for regression tests. They check that with each new updating the behavior of the tested code section has not changed. It means that the existing functionality has not broken its running. But one of the specific features of a company's IS are frequent changes in most components, and especially in the layer of business logic. The generated tests will lose their relevance within a month, and a significant part of the running time will have to be devoted to removing outdated tests. Moreover, such small unit tests help to test elementary blocks of code of the limited size. The tasks assigned to the created test on the automation solution relate to examining the correctness of the business processes operation, which can be performed both in small sections of code and within large sequences of actions covering several modules and their interaction.

In this regard, the black box testing was considered as an alternative one [10]. It is more suitable for solving the assigned tasks, since tests written in accordance with this methodology will examine the correctness of business processes, without being tied to a constantly changing implementation. Products such as MSpec, SpecFlow, IBM Rational Tau, Rhapsody Auto Test Generator (ATG), and Rhapsody Test Conductor Add-on have been evaluated. The evaluation concluded that there are many projects on the market that can help developers reduce the time to create tests and improve their quality. However, the specificity of the company's IS does not allow the application of these products for one of the following reasons:

- A significant redesign of the code base is required to meet the requirements of the generator code for the general architecture of the application
- A generator code specializes in another type of application and, as a result, it is not able to test the most important part of the business logic
- Generated tests are not designed for frequent changes in the application code, they lead to a large number of false responses
- The generated tests require significant improvements from the developer

An ideal general-purpose tool for cogeneration of automatically generating tests has not been identified, as it must be closely tied to the application code, its architecture and key classes, methods in order to ensure high quality of the test. This conclusion is not unique, since in the corporate segment of software development, most of the products are based on their own, closed architecture with non-standard, specific solutions [11, 12]. Nevertheless, the study of the existing products is of practical application for specific automatically generating tests, since after studying various approaches with their peculiar advantages and disadvantages. One can make an informed decision about the requirements for his own solution, a form of its interaction with the user and choose the optimal way to implement it.

## 2. Materials and methods

The main goal that a company is going to achieve thanks to the developed methodology for automating the process of automatically generating software tests is to reduce its costs for developing new automated tests that can examine the correctness of the business logic of various algorithms for processing user's data. It means that the developed methodology does not have to be a universal tool that helps to create tests for all components of the unified information system. It greatly simplifies the possible implementation of this methodology and narrows its capabilities, but at the same time it helps to get a better product at the expense of reasonable costs.

A typical automatically generating tests by the developed tool, is described according to the following scenario:

- Create a stock portfolio and related entities (for example, a legal entity to which the portfolio belongs)
- Carry out a verified transaction (for example, buying shares on the stock exchange)
- Calculate portfolio value
- Compare the cost obtained in step 3 with the expected value

Moreover, in addition to calculating the values of stock portfolios, the duties of a specialized depository include checking the correctness of the actions performed by the client. Thus, the second typical scenario is described by the following algorithm:

- Create a portfolio of stocks and related entities
- Attempt to perform the verified operation
- Make sure that an operation carried out in step 2 failed with an error

However, while developing a methodology for the automatically generating tests, it should be borne in mind that its implementation can have both a positive effect in the form of optimizing the costs of creating new tests, and a negative effect in the form of additional costs for maintaining automatically created tests [13, 14]. Therefore, it is necessary to note the quality requirements of the implemented tests while setting the requirements for the future methodology. Two significant aspects of quality have been highlighted here.

The first aspect is test detailing. Overly detailed tests can validate a lot of internal intermediate steps of data processing, thereby limiting developers' implementation capabilities. Such tests will require duplicating these changes in the test code when making changes to the main code. It will reduce the development rate of the main business logic. The introduction of such tests will significantly decrease the overall rate of developing innovations with a high variability of business requirements and data processing algorithms in the company's IS. The other extreme of low-quality tests is that they are not able to catch bugs in the code under test. This behavior can be demonstrated by tests with too preview quality or tests that are divorced from the real conditions of using the tested algorithms and are tied to the test environment. There is practically no benefit from such tests; they only bring harm due to the creation of an incorrect idea of the level of code coverage by tests.

The second aspect is the maintainability of the tests. Automatically generating tests are also code that, as the main business logic code, can be changed over time for objective reasons. For example, if a client specified in the new requirements that the commission for transferring funds should not be 10%, but 5%, this should be reflected in the test code as well, since the expected final state of the system changes along with the change in requirements. Therefore, the test code should be as clear, readable, and easily modifiable as the main application code.

Another key factor that must be taken into account at developing the implementation of this methodology is its integrability into IS. This system is a set of 20 unrelated simple modules, several finite modules that collect the functionality of simple modules, and kernel modules of the system, i.e., ten closely related assemblies. NETFramework. All other elements of the system depend on them. Such an architecture makes it relatively easy and independent to develop and maintain each separate module that implements its specific task. As a result, any module can be easily expanded without the need to change other system's components.

Taking into account the above information, additional requirements are formed for the developed methodology. First, it should be as easily expandable as a system in whole. Second, it should be borne in mind when developing a methodology, that its implementation will run in a multi-module system environment on the. NETFramework platform. The formalized requirements for the developed tool are stated as it should allow automatically generating tests of business logic at a lower cost, but at the same time the quality of automatically generating tests should correspond to the quality of the existing tests, i.e., they should evaluate the correctness of algorithms for processing user's data without details of the implementation of these algorithms. But a test code should be readable and easily modified.

## 3. Results and discussion

In order to automate any process, it is first necessary to formalize this process in the form of an algorithm, a sequence of actions to achieve the desired result. In order to automate the development of tests, it is necessary to formalize the questions that the developer answers when creating a test manually. However, it is the main disadvantage of the approach to developing tests by developers of the code, if a developer misunderstands the expected behavior of the system, he will make a mistake twice while the implementation of the algorithm and while the implementation of the test that examines this algorithm. In the case of automatically generating tests, it becomes much more difficult to answer these questions.

Taking into account the results of evaluating software test automation solutions available on the market, we considered two approaches to solving this problem.

The first approach is code generation of tests based on documentation. This approach shifts the responsibility for the quality control of the IS to the personnel who will be involved in the preparation of the documentation on the basis of which the automated tests will be generated. Definitely, it has some advantages, reducing the influence of the human factor on the quality of final tests, but at the same time, it carries with it some disadvantages, namely:

- Code generators consume not a textual description of an algorithm as input data, but its structured description in a specialized format. It means that the implementation of such a solution will require additional training of personnel not only in the culture of writing high-quality tests, but also in the study of the appropriate format for describing algorithms.
- Code generators are not able to generate the IS-integrated automatically generating test code. It is necessary to spend resources on adapting code generators for code, templates and IS interfaces to implement them.

The second approach is code generation of tests based on recorded user's actions. At the same time, the quality of tests remains at the same level, since the tests creation can no longer be done by developers, but by analysts or business users. Such a disadvantage as the need to write a code generator in this approach is also present; however, code generators at the input already consume their own format for representing recorded user's actions. The main advantage of user's actions recording is that, no additional study of the formats for presenting data processing algorithms is required to create a test; a user continues to work in a familiar system interface.

So, the most convenient way for the automatically generating tests is to generate tests based on recorded user's actions. Our own development, in contrast to the application of ready-made solutions, helps to integrate the code generator into the IS tightly, providing users with a convenient interface for creating tests and obtaining readable, easily maintainable tests using the interfaces and development patterns familiar to developers.

A typical scenario of user's interaction with a system to record a test is as follows:

- Click on the "Start test recording" button

- Perform manipulations in the system that reproduce the tested behavior
- Create checkpoints that examines the correctness of the performed actions
- Click on the "End test recording" button

Thus, people specializing in the company's business processes will be able to control the correctness of the IS operation directly, without the participation of an intermediary of a programmer. As a result, a source code of the recorded test should be generated. Definitely, it is not enough to have only the implemented methodology for the successful implementation of any methodology in a project. The state of the system itself, where new functionality is being integrated, is also an important factor in success, including the possibility of its expansion or change. These own developments help to design a solution so that the requirements are met with minimal effort. But it also has a drawback, i.e., the need to modify the company's infrastructure while the implementation. So, the developed methodology for organizing testing will require compliance with the following parameters (components) of the IT infrastructure for software quality control:

- Application of an extensible, open to testing IS architecture
- Development of the code generator integrated into the IS for automatically generating tests based on user's actions
- Conducting continuous automatic testing of all changes made to the codebase
- Training of users involved in the process of the automated tests development, the basic principles of software testing
- Organization of the delivery process of tests generated on the basis of user's actions to general test environment

Remind that the IS of the company is an expandable software package that will help to integrate the developed module for the automated tests development into several points of interaction between the system core and other components, and thereby track and record the necessary user's actions. Moreover, a native library of auxiliary classes will help to generate concise, understandable code that can be easily read and maintained in future. The composition and structure of the module being developed is presented in Figure 1:

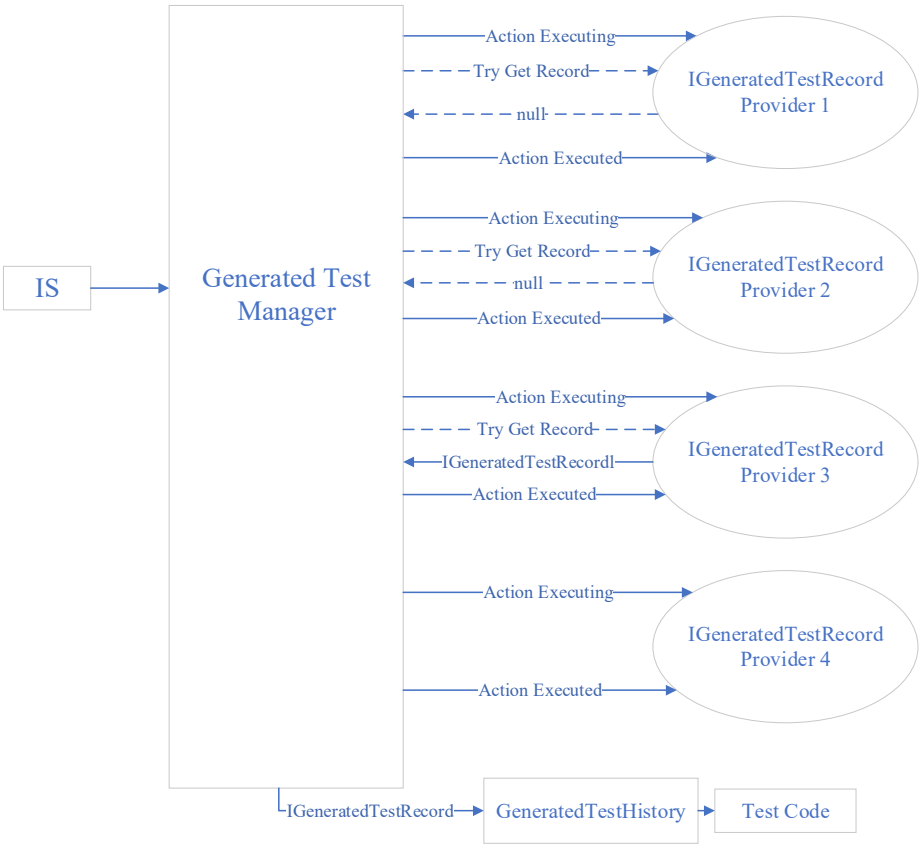


**Figure 1:** Architecture of the test module under development

The core of the module is a GeneratedTestManager class. It performs several functions: it is an entry point to the module; on/off test recording; manages the storage of records for the current test; synchronizes threads; defines an order of providers' work applying topological sorting; launches the work of providers; ensures the safe operation of the module in the IS. The IGeneratedTestRecordProvider interface, i.e., a provider of generated test records, has a defining value in the module. Implementations of this interface are involved in processing the fulfillment of all IS actions. We propose to solve the problem of interaction between loosely coupled or even unrelated providers by using topological sorting when resolving the priority of providers. The next logical solution is to take the results of the providers' work into a separate entity. It will help to collect the generated records together and process them as a whole, taking into account their impact on each other. Also, the IGeneratedTestRecord interface defines the AdditionalActions field, which contains a collection of additional, side-effects of the record. These additional steps will help to develop the context required to execute the master record. They will also be used to explicitly express side effects or dependencies.

Corcoranmodel was chosen to evaluate the reliability of the developed module, [15], according to which the coefficient (R) was calculated. In total, 90 tests were carried out during the testing process, 79 of them were unsuccessful, i.e., they did not reveal the existing problems. The value of the R coefficient was 0.89. Thus, with 89% confidence, it can be argued that the developed module will work stably, without failures. Such indicators are a satisfactory result considering that the operation of this software tool does not affect the stability of the surrounding system as a whole and does not have access to real customer data,

The implementation of the new module made it possible to optimize the company's costs in two directions at once. First, there is a partial transfer of responsibility for tests from developers to analysts. This is possible due to the fact that there is no necessity to know the C # programming language to develop tests. It is possible to do these applying elements of the web interface. Second, the optimization of the time that it takes to create a test by a developer due to the fact that it is much easier and faster to perform actions in the web interface than to search, configure and use the corresponding classes in the code.

The economic effect of the methodology implementation is represented by the change in the costs of supporting and developing the IS, since the development of the test generation module and its implementation will not increase the company's revenues. Reducing the labor costs for the automated tests development allowed reallocating the cost structure between the analytics and development departments. The conducted analysis shows that applying the new methodology, the company can save about 700 thousand rubles a year, i.e., 25% of the total costs for the automatically generating tests Table 1.

**Table 1**
Evaluation results of economic efficiency

| | Base case costs | Project version costs | Absolute cost change | Cost change factor | Cost change index |
|---|---|---|---|---|---|
| Labor intensity, hours | 3 900 | 2 950 | 950 | 24% | 1.32 |
| Cost, rub. | 2 850 000 | 2 150 000 | 700 000 | 25% | 1.33 |

Also, indicators of economic efficiency were significantly influenced by the capital cost of development, testing, debugging and implementation of the software tool. These costs represent a negligible share of the company's total costs and are recouped in just 41 working days. It makes the application of the developed testing module cost-efficient even in a short term.

## 4. Summary

The developed methodology of the code generation for automated tests implies the participation of IS users in the process of IS testing. This approach implies a high level of trust in users. For companies where the users of the system are internal employees, this limitation is not an obstacle to the implementation of code generation for automated tests based on user's actions, but it may require additional costs for the company to train employees in the culture of tests creating. It is important to understand that not all user actions in the system check the correctness of the application's business logic, and low-quality tests will inevitably lead to a negative effect due to the presence of a large number of false positive or false negative test results.

Also, a company, when implementing the developed methodology for the automated tests development based on user's actions, should establish processes for delivering the developed tests to test environment. A typical organization of the process of the new automatically generating tests the introduction of the automated test code into the general test base, either by a developer or by a tester. The automatically generating tests through the user interface provides an opportunity to exclude manual work with the test base and automate the test delivery process. Modern version of control systems and repositories provide all the necessary interfaces to implement this integration.

Thus, a technical part of the tool implementation does not limit a company in the range of potential opportunities, but the feasibility of developing such a tool and the set of functions provided should be determined by a company itself based on the previously considered factors, such as the level of user qualifications, the availability and quality of automated testing systems, openness to expansion of the information system itself.

At the same time, an effective IT infrastructure for quality control is not limited only to the application of the developed methodology for automatically generating test code. It also touches upon the issues of designing the architecture of the software under test, it includes some measures for training employees and organizing automated testing processes. The application of the developed methodology is advisable when the existing IT infrastructure of the company already meets at least some of the previously listed requirements, but its implementation can help the company significantly reduce the costs of the process of creating tests.

The description of the process of developing a code generation module for automatic tests for ISs is given as a part of the implementation of this methodology in the company. The given architecture of the software tool is not the only the correct one, but it helps to demonstrate that in the presence of the extensible architecture of the system itself, the development of a new module that automates the process for automatically generating tests is a relatively simple task. It helps due to the low costs of developing a new tool, to reduce significantly maintenance costs of the existing processes for the automatically generating tests.

## 5. References


[1] D. Rana, M. Malhotra, Analyzing the Role of Risk Mitigation and Monitoring in Software Development, Problems, threats in software development life cycle and their analysis (2018) 61–82. doi: 10.4018/978-1-5225-6029-6.ch005.

[2] V. Buvaltseva, V. Chechin, Development of the investor institution in Russia as a basic participant of securities market, Voprosy Ekonomiki (2015) 61–75. doi: 10.32609/0042-8736-2015-3-61-75.

[3] A. Shi, P. Zhao, D. Marinov, Understanding and Improving Regression Test Selection in Continuous Integration, Proceedings - International Symposium on Software Reliability Engineering, ISSRE, (2019), pp. 228–238. doi. 10.1109/ISSRE.2019.00031.

[4] D. Marijan, A. Gotlieb, M. Liaaen, A learning algorithm for optimizing continuous integration development and testing practice, Software - Practice and Experience 49(2) (2019) 192–213. doi: 10.1002/spe.2661.

[5] M. Erder, P. Pureur, Chapter 5 - Continuous Architecture and Continuous Delivery, ed M. Erder and P. Pureur, Boston: Morgan Kaufmann, 2016, pp 103–129.

[6] Y. Huang, P. Mishra,Hardware IP Security and Trust, Test generation for detection of malicious parametric variations, 2017, pp. 325-340. doi: 10.1007/978-3-319-49025-0_14.

[7] W. Yan, D. Fontaine, J.A. Chandy, L. Michel, A design flow with integrated verification of requirements and faults in safety-critical systems, 12th System of Systems Engineering Conference, 2017. doi: 10.1109/SYSOSE.2017.7994955.
[8] I. Essebaa, S. Chantit, M. Ramdani, Model-based testing from model driven architecture: A novel approach for automatic test cases generation, Lecture Notes in Electrical Engineering, volume 684 LNEE, 2020, pp. 600–609. doi: 10.1007/978-3-030-53187-4_66.
[9] S. Kamalakar, S.H. Edwards, T.M. Dao, Automatically generating tests from natural language descriptions of software behavior, ENASE 2013 - Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (2013) 238–245.
[10] A. Paul, O. Jeff, Introduction to Software Testing, Cambridge University Press, 2017.
[11] D. Badampudi, C. Wohlin, K. Petersen, Software component decision-making: In-house, OSS, COTS or outsourcing - A systematic literature review, Journal of Systems and Software 121 (2016) 105–24. doi: 10.1016/j.jss.2016.07.027.
[12] T. Shibata, M. Yano, F. Kodama, Empirical analysis of evolution of product architecture, Research Policy 34(1) (2005) 13–31. doi: 10.1016/j.respol.2004.09.011.
[13] G. Rudžionienė, S. Packevičius, E. Bareiša, Directed multi-target search based unit tests generation,Communications in Computer and Information Science, volume 1078, CCIS, 2019, pp. 90-109. doi: 10.1007/978-3-030-30275-7_8.
[14] T. Huertas, C. Quesada-López, A. Martínez, Using Model-Based Testing to Reduce Test Automation Technical Debt: An Industrial ExperienceReport, Advances in Intelligent Systems and Computing, volume 918, 2019, pp. 220-229. doi: 10.1007/978-3-030-11890-7_22.
[15] M. Xie, Software Reliability Models - Past, Present and Future, Birkhäuser, Boston, MA, 2000. Doi: 10.1007/978-1-4612-1384-0_21.