ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ

Chelyabinsk Physical and Mathematical Journal. 2019. Vol. 4, iss. 2. P. 241–249.

DOI: 10.24411/2500-0101-2019-14210

A REGRESSION TESTING WITH SEMI-AUTOMATIC TEST SELECTION FOR AUDITING OF IMS DATABASE

A.N. Ruchay

Chelyabinsk state university, Chelyabinsk, Russia
ran@csu.ru

This work aims to develop a regression testing framework with a semi-automatic test selection. As opposed to previous works it offers a manual combined approach to the development of regression testing with a semi-automatic test selection with according to various important factors such as the execution time, the required number of the test executions, the manual priority of the tests, a result of the previous tests, functionalities of the tests. It allows a more detailed manual configuration of the regression test sequence.

Keywords: *software testing, regression testing, debugging, keyword driven testing, data driven testing, IMS database, test selection.*

1. Introduction

Modern software projects evolve rapidly as developers add new features, fix bugs, or perform refactoring [1–3]. To ensure that a project evolution does not break existing functionality, developers commonly perform the regression testing. However, frequent re-running of full regression test suites can be extremely time consuming. Some test suites require weeks to run, but waiting even a few minutes for test results can be detrimental to developers' workflow. In addition to reducing developers' productivity, slow regression testing can consume a lot of computing resources. As a result, a large body of research has been dedicated to reducing the costs of regression testing, using approaches such as the regression test selection [4; 5], the regression test-suite reduction [6; 7], the regression test-case prioritization [8; 9], and the test parallelization. In article [10] a thorough survey of regression testing approaches is provided.

The regression testing approach we present here differs from other approaches in that it uses a semi-automatic test selection process to prioritize test cases. This prioritization is based on various criteria such as the execution time, the required number of the test executions, the manual priority of the tests, a result of the previous tests, functionalities of the tests. This approach allows a more detailed manual configuration of the regression test sequence.

The paper is organized as follows. Section 2 provides information about regression testing of IMS database auditing system. Section 3 describes various automated testing and test automation framework approaches. Also in Section 3 framework structure is described with web interface. Section 4 is devoted to examining a regression test selection approach that speeds up the regression testing. Proposed regression testing with the semi-automatic test selection is presented in Section 4. Finally, section 5 presents our conclusions.

2. Regression testing of an audit system for an IMS database

IBM Information Management System (IMS) is a joint hierarchical database and the information management system with extensive transaction processing capabilities [11–13]. IMS hierarchical databases provide numerous advantages: they are a proven technology that has been in place for many years, their hierarchy (the tree structure) is easy to use, they are very good at modeling hierarchical data relationships, they manage over 15 million gigabytes of production data and server over 200 million users processing, and they can support in excess of 22000 transactions per second. The main disadvantage of IMS hierarchical databases is their complexity, both in terms of rules and many-to-many data relationships. This complexity has caused IMS development to remain very slow and cumbersome [11–13].

An audit system for an IMS database on z/OS is an auditing tool that collects and correlates data access information from IMS online regions, IMS batch jobs, IMS archived log data sets, and SMF records to produce a comprehensive view of business activity that occurs within one or more IMS environments. z/OS is an operating system for IBM mainframes, produced by IBM. An audit system for an IMS database helps auditors to determine who read or updated a particular IMS database and its associated data sets, what mechanism was used to perform that action, and when the access took place.

An audit system for an IMS database provides a comprehensive auditing solution that reduces the cost of compliance through automation, centralization, and segregation of duties. Automation simplifies the collection of audit data, reduces manual efforts, increases productivity, achieves more thorough audits, and reduces the cost of auditing. An audit system for an IMS database centralizes and correlates information from many systems and data sources to provide a coherent view of IMS audit data, create GUI and batch reports for examining the data, and support internal and external auditors. Segregation of duties provides audit data integrity, removes opportunity for data tampering, provides that auditors are no longer dependent on developers or database administrators to set up or gather the audit information required, frees up DBAs to perform their own duties and allows auditors to run audit reports independently of the DBAs.

To test the audit system for IMS databases, we used 50 online z/OS jobs and 50 batch z/OS jobs with 1000 transactions per second, 10 stress testing scripts with 100 000 transactions per second, 1000 IMS LOG transactions and 3000 SMF transactions to simulate data access information from an IMS database. Our tests included 120 automated tests, 350 test cases, 4 versions of an audit system for an IMS database, 4 versions of an IMS databases, and 3 z/OS subsytems.

Constant regression testing (general testing, security testing, performance testing, stress testing) is required to ensure that the addition of new features, fixes, or refactorings does not break existing functionality. Yet frequent re-running of full regression test suites is extremely time consuming. This testing might require weeks to run and can be detrimental to developer workflow. In addition to reducing developer productivity, slow regression testing can consume significant computing resources. Our regression testing requires about 3 months to run all tests manually. The main goal was to create a test automation framework that decreases regression testing time.

3. Test automation framework

Automation testing is the execution of test cases in an automated manner, without manual intervention [14]. Automation testing includes various activities such as test generation, reporting of test execution results, and test management. This method of testing has evolved over time as five generations [15; 16]: record and replay, custom scripts, data-driven testing, keyword-driven testing; and process-driven testing.

A test automation framework is an integrated system that sets the rules by which a product undergoes automated tested. The test automation framework integrates function libraries, test data sources, object details, and various reusable modules. It provides a basis for testing and simplifies the automation effort. The most common categories of test automation frameworks include [15; 16]: module-based testing framework, data-driven testing framework, keyword-driven testing framework, and hybrid-testing framework.

A module-based testing framework is a basic type of automation framework. A module-based testing framework uses test scripts that match product functionality and correspond to specific modules of the application. These test scripts are used in a hierarchical manner, to build large test cases. A module-based testing framework assigns independent test scripts to specific software components, modules, or functions.

In data-driven testing frameworks, test data is separated from test scripts, and test results are returned against the test data. When tests use varied sets of input and output test data, data-driven testing frameworks are more efficient and easier to manage.

In keyword-driven testing frameworks, keywords are written such that they are equal to unit-level functionality. A keyword-driven testing framework is application-independent and uses data table methods and keywords to perform the actions.

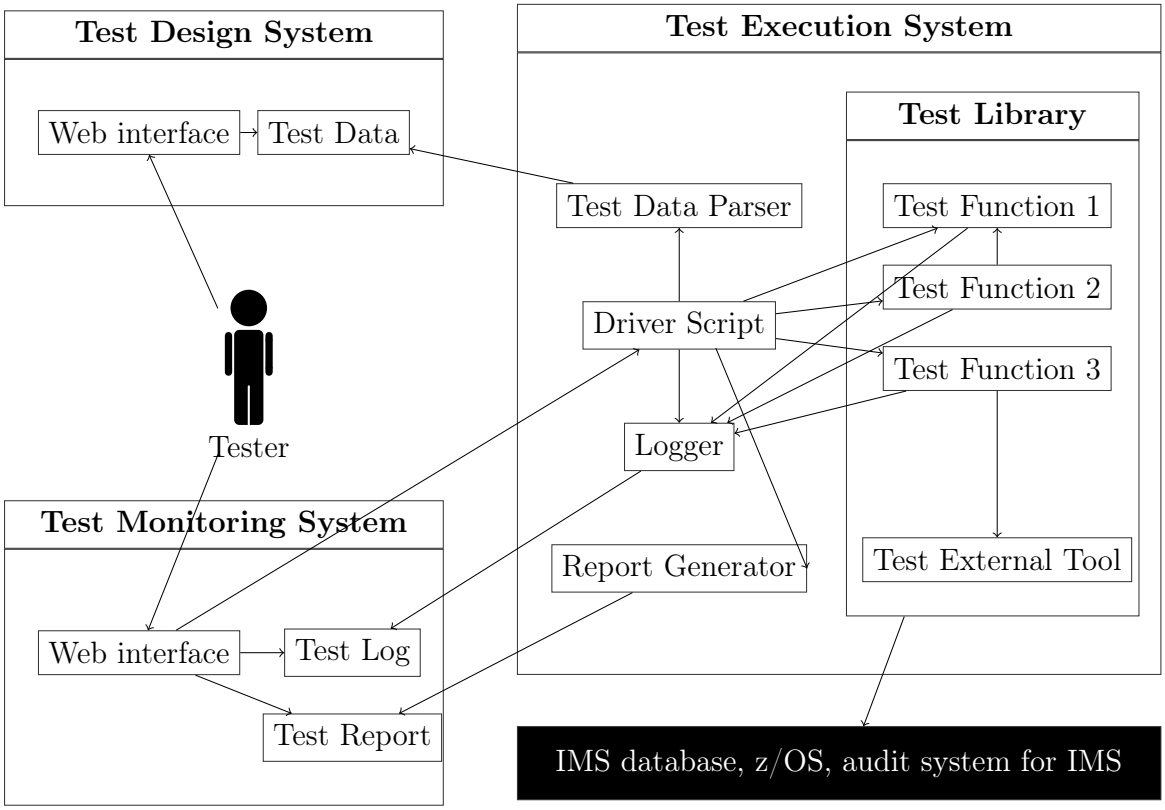
Hybrid-testing frameworks incorporate features such as script modularity, data-driven framework, and functional decomposition. A hybrid-testing framework can be more complex to set up initially than other frameworks. However, hybrid-testing frameworks can provide the most flexibility if they are carefully evaluated and implemented.

The test automation framework structure that we used consists of a test design system, a test monitoring system, and a test execution system [17]. Figure illustrates the framework and shows how components work together.

The web interface we used with our test automation framework was developed with Qooxdoo, an open source Ajax web application framework. It is a client-side, server-agnostic solution that includes support for professional JavaScript development [18], a graphical user interface (GUI) toolkit, and high-level client-server communication.

The Java libraries that we used for developing our test automation framework include: S3270 for integration TN3270 system on z/OS, Selenium WebDriver for browser instance, Ganymed SSH-2 for implementing the SSH-2 protocol, Apache Commons Net for implementing the client side of many basic Internet protocols, Apache FreeMarker for generating text output based on templates and changing data, Jsoup for working with real-world HTML, Gson for serializing and deserializing Java objects to JSON, Apache log4j for logging, httpclient for implementing the client side of the most recent HTTP standards and recommendations.

Test design system is used for creating new test cases and editing existing ones. It is easy to use with minimal training and without programming skills. The created test data can be stored in files. A simple solution is using an existing tool like JSON editor as a test design system [19]. JSON is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is the



Framework components

most common language-independent data format used for asynchronous browser/server communication. JSON offers a number of advantages over XML: it can be parsed faster, it is more compact, it is easier to work with in some languages, and (when formatted) is generally easier to read.

Test monitoring system is used for controlling test execution and checking test results. It has the following capabilities: starting test execution manually; starting test execution automatically after some specified event; starting test execution automatically at specified time; stopping test execution; monitoring test execution while tests are running; viewing test logs while tests are running and afterwards; viewing test report; changing logging level; selecting regression tests; configuring tests. The test monitoring solution is using web interface for starting and stopping test execution and creating logs and reports in HTML and JSON format.

Test execution system is the core of the framework. Its main components are driver scripts, the test library, the test data parser and other utilities like log and report generator. Test execution is controlled by driver scripts which are started using the test monitoring system. Driver scripts are pretty short and simple because they mainly use services provided by test libraries and other reusable modules. Test libraries contain functions and modules which are used in testing or in activities supporting it. Testing is mainly interacting with the tested system and checking that it behaves correctly. Functions in the test library can do their tasks independently but they can also use other functions or even external tools. For example, framework TN3270 is used to implement work with the z/OS and IMS database, framework Selenium [20] and REST API — web interface of audit system for IMS database. The test data parser is intended to provide test data easily to driver scripts by JSON parser in Java. Its task is processing the test data and returning it to the driver script in easy to use test data containers.

Our test automation framework was built with several reusable modules and function libraries that are developed with the following features:

- Maintainability — framework extensively reduces maintenance effort;
- Re-usability — test cases and library functions can be reused;
- Manageability — effective test design, traceability and execution;
- Accessibility — easy to develop, design, modify and debug test cases while executing;
- Availability — allows to schedule automation execution;
- Reliability — advanced error handling and scenario recovery;
- Flexibility — framework independent of system under test;
- Measurability — customizable reporting of test results ensure the quality output.

High level requirements for large scale test automation frameworks are automatic test execution, ease of use and maintainability. Our framework satisfies the following requirements [15; 16]:

- High Level Requirements. The framework executes test cases automatically, is easy for using without programming skills, simple for support, doesn't need permanent control while executing tests. It allows to run tests manually or schedule automatic execution at predefined time or after certain events. Non-fatal errors caused by test environment are handled and reported without stopping test execution. Test results are verified. Every executed test case is marked as Passed or Failed and failed test cases have a short error message. Test execution is logged using different configurable logging levels. Test report is created and published automatically.
- Ease of Use. The framework uses keyword driven approach, supports creating user keywords. It provides functionality that selects and groups tests for particular task.
- Maintainability. The framework is modular and has coding and naming conventions. It is implemented using high level scripting languages. The testware in the framework is under version control (Git) and is adequately documented.

A regression testing requires 3 weeks to run by our test automation framework instead of 3 months of manual testing.

4. Proposed regression testing with semi-automatic test selection

A widely-used method to speed regression testing is referred to as Regression Test Selection (RTS) [3]. RTS reduces testing time by re-running only the tests that are affected by a code change. RTS is a reliable technique if it tests all of the scenarios that might be affected by the code change. If any scenarious that are affected by the code change are not tested, regressions might be missed. Prior research on RTS can be broadly split into dynamic and static techniques.

The lack of practical RTS tools leaves two options for developers: they must either automatically re-run all tests or perform manual test selection (manual RTS). Automatically re-running all tests is a safe approach, but it can be imprecise and inefficient. Manual RTS, in contrast, can be unsafe and imprecise for several reasons. When using manual RTS, developers might select too few tests and, as a result, exclude tests that might expose differences due to code changes. They also might select too many tests and thus waste time. Therefore, developers could benefit from an automated RTS technique that works in practice. It has not been established how manual RTS

practices used by developers compare with the automated RTS techniques proposed in the literature.

There are the following technique for regression test selection [21]:

1. Retest-All Technique: This technique tests all test cases that were previously run during testing phase. This technique is very expensive because regression test suites are costly to execute.
2. Random/Ad-Hoc Technique: When using this technique, testers rely on their previous experiences and knowledge to select which test cases need to be rerun. This can include selecting a percentage of test cases randomly.
3. Dataflow Technique: This technique is a coverage-based regression test selection technique that selects test cases that exercise data interactions that have been affected by modifications.
4. Safe Technique: This technique eliminates the test cases that are unlikely to reveal faults. This technique routinely selects almost all test cases when changes were made to the programs. This technique does not focus on criteria of coverage but select all those test cases that produce different output with a modified program as compared to its original version.
5. Hybrid Technique: This technique includes Test Case Prioritization and Regression Test Minimization.

In contrast to previous works [8], we offer a combined approach to select a sequence of tests that uses simple criteria and rules such as total efforts, execution time, deploying time, and so on. This requires manual and automated settings.

The basic criteria and rules for semi-automatic regression test selection in terms of our approach include:

1. Total efforts, the number of required tests.
2. Required time for tests, total time execution.
3. Required number of particular test executions.
4. Availability of necessary IMS databases, z/OS subsystems.
5. Priority of tests.
6. Results of previous tests.

Our proposed regression testing with semi-automatic test selection prioritizes test cases based on highest priority according to above basic criteria and rules. For any test case t_i, define set of attributes $a_i \subseteq A$. Let A represent the set of the functionalities of auditing IMS database that is manually specified by tester.

Our proposed regression testing with semi-automatic test selection prioritizes test cases based on highest priority according to above basic criteria and rules. Consider a test suite T containing n test cases, $\{t_1, t_2, \ldots, t_n\}$. For any test case t_i, define set of attributes $a_i \subseteq A$. Let A represent the set of the functionalities of auditing an IMS database that is manually specified by tester. Functionalities is defined by 10 types of filtering and 7 types of information for auditing an IMS database.

For any test case t_i, let $p_i = \prod_{j=1}^{k} t_i^j$ denote priority of using test case, where t_i^j — assessment of factor of using test case t_i with the criteria j.

Here are the criteria j for any test case t_i in proposed regression testing with semi-automatic test selection:

- $t^1 = \{0, 1\}$ — factor of availability of necessary z/OS subsystem, version of an audit system for an IMS database, version of an IMS databases. $t^1 = 1$ is available, and $t^1 = 0$ is not available;

- $t^2 = [1, 10]$ — priority of test. Tester sets this factor for each test case t_i. For example, $t^2 = 10$ for high priority;
- $t^3 = [1, d]$ — factor of result of previous tests. This factor is calculated for all previous test executions. d is a number of program failures revealed by the test cases t_i;
- $t^4 = [1, 20]$ — required number of test executions. Tester sets this factor for each test cases t_i. For example, $t^4 = 10$ for jobs with high risk of failure;
- $t^5 = [0, 10]$ — factor of specification requirements based on information from developers;
- $t^6 = [1, 10]$ — factor of execution time of tests. For example, $t^6 = 1$ for long and $t^6 = 10$ for short test case.

The following algorithm prioritizes the use of test cases in our proposed regression testing with semi-automatic test selection:

1. Specify z/OS subsystem, version of audit system for IMS database on z/OS, version of IMS databases.
2. Specify criteria t^j (some of them should be specified manually).
3. Specify subset T' of test cases in accordance with subset A manually (not necessary for automatic test selection).
4. Estimate priority of using test cases $\{p_1, \ldots, p_n\}$ by evaluating all criteria t^j.
5. Order (sort) test cases t_i in subset T' according to the value of p_i.

Regression testing with our approach requires one day for smoke testing and one week for detailed regression testing to meet the requirements of our management and customers. Instead of focusing only on reducing the number of selected tests, our approach provides a significant reduction in regression testing time and balances the time needed for the analysis, collection, and execution phases.

5. Conclusion

Regression testing is important for verifying that software changes do not break previously working functionality. However, regression testing is costly because it runs many tests for many revisions. Although RTS, proposed more than three decades ago, provides a promising approach to speed-up regression testing, no RTS technique has been widely adopted in practice due to efficiency and safety issues.

Most developers perform manual RTS. They select tests in ad-hoc ways, potentially missing bugs or wasting time [5]. The approach we presented here proposes regression testing with semi-automatic test selection. This approach prioritizes test cases based on priority according to some criterion.

The regression testing approach we present here differs from other approaches in that it uses a semi-automatic test selection process to prioritize test cases. This prioritization is based on various criteria such as execution time, required number of test executions, manual priority of tests, result of previous tests, functionality of tests. This approach allows a more detailed manual configuration of the regression test sequence.

Our approach provides a substantial reduction in the time required for regression testing and balances the time required for analysis, collection, and the execution instead of focusing solely on reducing the number of selected tests.

Our study evaluated the feasibility of the presented test automation framework for regression testing of an IMS database auditing system. The overall assessment was positive and the framework was declared valid. The presented test automation framework satisfies important lower-level requirements in terms of ease-of-use and maintainability.

References

1. **Tulpule N.** Strategies for testing client-server interactions in mobile applications: How to move fast and not break things. *Proceedings of the ACM Workshop on Mobile Development Lifecycle*, 2013, pp. 19–20.
2. **Jensen C.S., Moller A., Su Z.** Server interface descriptions for automated testing of javascript web applications. *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 510–520.
3. **Legunsen O., Hariri F., Shi A., Lu Y., Zhang L., Marinov D.** An extensive study of static regression test selection in modern software evolution. *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 583–594.
4. **Rothermel G., Harrold M.J.** A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 1997, vol. 6, no. 2, pp. 173–210.
5. **Gligoric M., Eloussi L., Marinov D.** Practical regression test selection with dynamic file dependencies. *Proceedings of the International Symposium on Software Testing and Analysis*, 2015, pp. 211–222.
6. **Zhong H., Zhang L., Mei H.** An experimental study of four typical test suite reduction techniques. *Information and Software Technology*, 2008, vol. 50, no. 6, pp. 534–546.
7. **Shi A., Yung T., Gyori A., Marinov D.** Comparing and combining test-suite reduction and regression test selection. *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 237–247.
8. **Hao D., Zhang L., Zhang L., Rothermel G., Mei H.** A unified test case prioritization approach. *ACM Transactions on Software Engineering and Methodology*, 2014, vol. 24, no. 2, pp. 10–31.
9. **Zhai K., Jiang B., Chan W.K.** Prioritizing test cases for regression testing of location-based services: Metrics, techniques, and case study. *IEEE Transactions on Services Computing*, 2014, vol. 7, no. 1, pp. 54–67.
10. **Yoo S., Harman M.** Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, 2012, vol. 22, no. 2, pp. 67–120.
11. **Burleson D.K.** *Inside the Database Object Model*. 1st ed. Boca Raton, CRC Press, Inc., 1998. 240 p.
12. **Long R., Harrington M., Hain R., Nicholls G.** *IMS Primer*. IBM Redbooks, 2000.
13. **Klein B., Long R.A., Blackman K.R., Goff D.L., Nathan S.P., Lanyi M.M., Wilson M.M., Butterweck J., Sherrill S.L.** *An Introduction to IMS: Your Complete Guide to IBM Information Management System*. 2nd ed. IBM Press, 2012.
14. **Rashmi, Bajpai N.** A keyword driven framework for testing web applications. *International Journal of Advanced Computer Science and Applications*, 2012, vol. 3, no. 3, pp. 8–14.
15. **Abhishek K., Kumar P., Sharad T.** Automation of Regression Analysis: Methodology and Approach. *Advances in Computer Science, Engeineering & Applications*, eds. D.C. Wyld et al. Berlin, Heidelberg, 2012. Pp. 481–487.
16. **Kumar P., Kavita D.** Automation framework for database testing. *5th International Conference on Recent Innovations in Science, Engineering and Management*, 2016, pp. 88–93.
17. **Laukkanen P.** *Data-driven and keyword-driven test automation frameworks*, Master's thesis. Helsinki, Helsinki University of Technology, 2006. 98 p.
18. **Artzi S., Dolby J., Jensen S.H., Moller A., Tip F.** A framework for automated testing of javascript web applications. *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 571–580.
19. **Sabev P., Grigorova K.** Manual to automated testing: An effort-based approach for determining the priority of software test automation. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2015, vol. 9, no. 12, pp. 2456–2462.

20. **Gojare S., Joshi R., Gaigaware D.** Analysis and design of selenium webdriver automation testing framework. *Procedia Computer Science*, 2015, vol. 50, pp. 341–346.
21. **Jyoti K.S.** A comparative study of five regression testing techniques: A survey. *International Journal of Scientific & Technology Research*, 2014, vol. 3, iss. 8, pp. 76–80.

Accepted article received 16.03.2019
Corrections received 05.05.2019

Челябинский физико-математический журнал. 2019. Т. 4, вып. 2. С. 241–249.

УДК 004.05 DOI: 10.24411/2500-0101-2019-14210

РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ С ПОЛУАВТОМАТИЧЕСКИМ ВЫБОРОМ ТЕСТОВ ДЛЯ АУДИТА БАЗЫ ДАННЫХ IMS

А. Н. Ручай

Челябинский государственный университет, Челябинск, Россия
ran@csu.ru

Работа направлена на разработку основы регрессионного тестирования с полуавтоматическим выбором тестов. В отличие от предыдущих работ предлагается ручной комбинированный подход к разработке регрессионного тестирования с полуавтоматическим выбором тестов с учётом различных важных факторов, таких как время выполнения, требуемое количество тестов, ручной приоритет тестов, результат предыдущих тестов, функциональные возможности тестов. Это позволяет более детально настроить последовательность тестов для регрессии.

Ключевые слова: *тестирование программного обеспечения, регрессионное тестирование, отладка, тестирование на основе ключевых команд, тестирование на основе данных, база данных IMS, выбор теста.*

Поступила в редакцию 16.03.2019
После переработки 05.05.2019

Сведения об авторе

Ручай Алексей Николаевич, кандидат физико-математических наук, доцент, заведующий кафедрой компьютерной безопасности и прикладной алгебры, Челябинский государственный университет, Челябинск, Россия; e-mail: ran@csu.ru.