LATVIJAS UNIVERSITĀTES RAKSTI. 2008, 733. sēj.:
DATORZINĀTNE UN INFORMĀCIJAS TEHNOLOĢIJAS 24.–37. lpp.

Application of Smart Technologies in Software Development: Automated Version Updating

Zane Bičevska, Jānis Bičevskis

Datorikas Institūts DIVI, A.Kalniņa str. 2-7, Rīga, Latvia

University of Latvia, Raiņa blvd. 19, Rīga, Latvia
Zane.Bicevska@di.lv, Janis.Bicevskis@lu.lv

Abstract. This paper proposes software version update solutions in compliance with smart technologies [1] - architectural designs and software components which using metainformation on the system and its operation requirements are able to solve efficiently the problems of software maintenance.

In order to ensure automated version update the authors propose several mutually independent mechanisms such as environment testing, software version update, automated data migration to the latest versions as well as automated self-testing of the installed version on internal consistency.

Based on experience in software development, distribution and introduction, the authors identify and describe particular framework principles for integration of automated function of software version update into produced software.

Keywords: Software engineering, Maintenance, Testing, Smart Technologies.

1 Introduction

The IT industry is characterized by multiplicity of infrastructure, applications and executable environments. Also, significant resources are required to adapt customized software for usage in multiple environments. The fierce competition in IT industry dictates fast appearance of high quality and innovative products on the market. Thus, the software developers encounter serious challenge – under significant time pressure to develop and deliver a software usable in multiple environment.

There are two approaches used in the praxis to ensure usability of developed and delivered software. In the first case requirements for software usability are defined, e.g. minimal requirements for hardware and version of operating system. However, this approach may cause problems such as mutually conflicting software versions. Additional difficulties arise if users are not ready or do not understand how to satisfy the defined requirements.

The other approach foresees that developers ensure usage of software in multiple environments and their combinations. The platform independence that foresees usage of software in various operating systems, on different data base management systems and using various browsers might require significant additional resources. In fact, all efforts to develop "universally applicable" software result in necessity to develop technical solutions for each of the environments.

Usage of universal algorithms and metadata only partially can unburden development of "universally applicable" software, for in result the executable code will remain platform dependent. Therefore, users of software are forced to encounter difficulties arising from software installing and updating – considerable time consumption and efforts are spent analyzing collision and problem causes, for consultations and for improving software to ensure its adequacy to the specific environment.

Consequently, the need arises for certain software development principles to ensure collision free (or reduced to minimum) software installation and download of software version updates. The following parts of the paper deal with smart technologies principle in essence – software should comprise features that support installing the latest versions without user assistance and ensuring the following:

1) automated analysis of the software compliance with environmental requirements
2) automated download and installation of the latest version of the software modules
3) automated personalization and individual data migration to the latest version
4) self-testing of the latest version, checking operating correctness of the information system on critical functionality
5) generation of back-up files and system renewal in case of failure

Even partial application of these principles (we are not aware of the cases when such principles were applied in full extent) has proven to be very useful.

2 Software Life Cycle Models and Smart Technologies

Over the years a number of discussions have been devoted to software development life cycle models [2] and analysis on strengths and weaknesses of linear and incremental models have been performed. A new approach (e.g. principles agile software development process) has been added to lengthy discussions. Nevertheless, the main attention in software life cycle models traditionally is being paid to software development, including requirement gathering (specification), design, implementation and testing. Less research is devoted to the system maintenance and operation despite the fact that these aspects take up the main part of the duration of a successful system.

In real life development of software is just the very first phase of software life cycle. The most time consuming phase of software life cycle is software maintenance including user support, software upgrade, and functionality extension services (e.g. MS Windows is being under development for more than 15 years!)

Every successful software solution has been used and improved for significantly longer time period that it was created for. A successfully developed system might be used for many years simultaneously modified, improved with new functions permanently satisfying occurring client needs. Thus, every time changes occur in software or operational environment, the issue of testing software's correctness becomes crucial.

Therefore, to ensure software reliability in long-term, the system already in its early development phases should comprise not only client defined functionality but also additional mechanisms to support usage, maintenance, and further development of software.

The smart technology is based on the idea about "smart" software that like living beings is able to "self-management". It means the software should be able to handle unpredictable events in unknown environments. A smart technology conform software should be able to deal with internal (related to internal structure and functioning of the system) and external (originated outside of the system) events adequately.

The targets set in the self-adaptive software [3, 4] partially overlap with the principles of the smart technologies. Self-adaptive software researchers are focusing on software ability to adapt itself to implementation environment. This report sets different targets: troubleshooting SW exploitation failures by applying automatic indication of possible failures and reporting them to staff. Implementation of this approach is more convenient to use in practice.

Smart technology oriented approach is illustrated in the succeeding figure (Fig.1). It demonstrates that the core functionality of software solution should be enhanced with several additional features supporting the usage, maintenance, and further development of software. These additional features called smart technologies are created in the process of software design and implementing similar to scaffolding in the construction process of a building. However unlike the building process the "scaffolding" of an information system is never taken down; it stays in the information system for its whole life time.

Relatively, these features are divided into two main groups:

1) External stability – ability to analyse environment consistency to its performance conditions

2) Internal stability – ability to check and maintain internal consistence

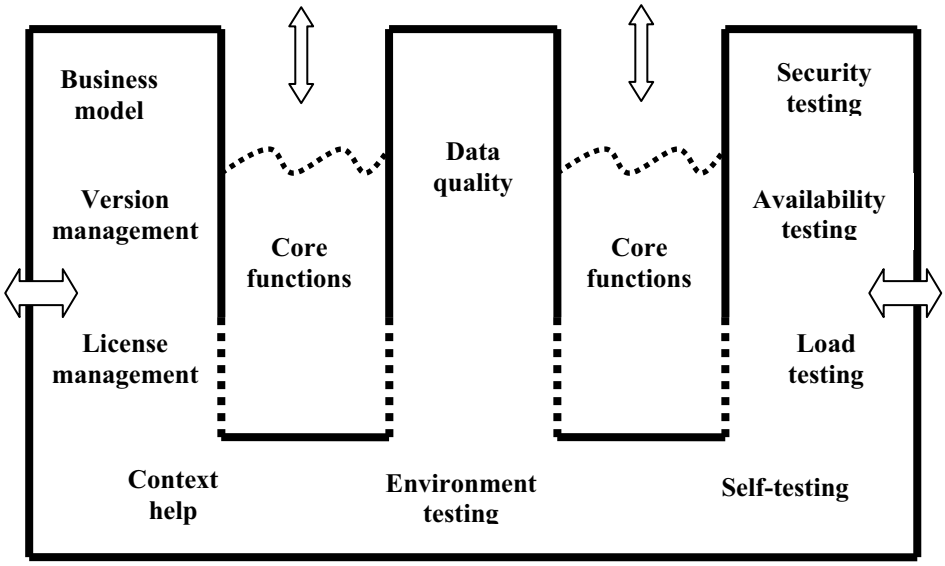

Fig. 1. Components of Smart Technology

Software fits to the principles of smart technology if it provides the following features:

1) Business model is incorporated into software
2) Version management – automatic updating of versions from the central server, including the adequate conversion of data structures and data
3) License management – automatic control of licence conditions, updating of standard and specific software according to them
4) Context help – integrated functionality of context sensitive help system
5) Environment testing - ability to analyse the external environment (for example, options of operating system and data base management system) and to adapt itself to the specific environment
6) Self-testing - ability to check the internal integrity by automatic execution of test cases in the productive environment and to inform users and developers about detected inconsistencies
7) Load testing - ability to provide monitoring of performance and load balancing
8) Availability testing – monitoring of system availability using agent technologies; ability to inform remote about the status of the software and additional components needed for a correct functioning
9) Security testing - monitoring of system security using agent technologies

10) Data quality monitoring - ability to check the completeness and integrity of data accumulated in the database

Detailed analysis of the features of smart technologies can serve as a subject of a separate research paper. This paper discusses only one of the smart technology features – automated version updating.

3 Automated Version Updating

It is a common practice in IT industry to pass (accept) the latest software version in the test environment before it is introduced in the production environment. However, in cases when system is used on many workstations and/or on several scattered servers, no warranty can be provided that this version will function problem-free for the following reasons:

1) each workstation and/or server contains individual data including e.g. personalization values, user rights, and even sophisticated definitions of processes
2) each workstation and/or server may be equipped with a different environment - different operating system, different versions of system database administration, file layout, regional settings, and other external systems operating parallely to the new software version

The first of proposed criteria is related to the field of delivering and installing (distributing) software. Information system equipped with smart technology is able to analyze the environment which it is put into from viewpoint of standard and specific parameters.

As standard parameters are supposed to be, for example, the operational system, the data base management system, browsers etc. used on the concrete server or workstation.

The specific parameters include checking of the evidence of previous (possibly damaged) software versions as well as evidence of other specific solutions on the workstation.

Automated software version updating includes both, external and internal stability criteria, since the environment analysis is performed before software version update and functionality self-testing after installation of version update.

Full cycle of automated version updating includes sequenced actuation of the following mechanisms in compliance with smart technology principles:

1) environment test
2) system back-up administration
3) downloading and installation of software version
4) data migration

5) self-testing
6) system renewal in cases of failure

3.1 Environment Test

The idea is taken from the real world which shows the ability of living beings to adjust themselves to specific conditions. Environment test includes the analysis of specific parameters by benchmarking their values and adapting functions to pre-defined decision-making flow.

In a similar way the software equipped with the smart technologies should be able to analyze versions and other configurations of the operating and data base management systems according to the requirements of the software, and react adequately to inconsistencies by generating necessary messages to users.

Although at the first glance, the task seems trivial, in fact the environment analysis (its compliance with the requirements) and subsequent decision-making (choice of the optimal way to adapt) are the most crucial tasks. In effect this analysis is much more complicated, because the version number and some of the configuration only partly determine the ability of the given application to fulfill their functions. An appropriate reaction depends on external factors, e.g. availability of internet connection to the necessary resources etc.

One of the solutions [5] aimed to simplify the analysis of an external environment is introduction of software application requirement passport that is created during the software development process in which the requirements against the environment are fixed: for operating system and other of its components, detailed on the level of object classes, of DLL's and of others - .ini-files, registry entries, location of files and folders, regional and language settings, workstation settings etc. The creation of the passport is an obligation for the developer; it happens by generation of the passport from the development environment in which the application is created and is able to work. The created passport is integrated into the software.

After implementation of the application in the production environment a module prepared by the developer compares the production environment parameters with the passport parameters (analysis of environment).

Differences between target environment parameters and values in passport may cause various reactions depending on adaptation mechanism. The most trivial reaction is when the user is notified about the differences and further steps necessary to correct environment configuration („Please, change regional settings").

High developed solutions are equipped with mechanisms aimed at collecting the missing components, e.g. automated component update is done from producer resources. These mechanisms are also able to reconfigure the environment according to the requirements [6].

Such automated mechanisms involved in environment adaptation also may seriously endanger operations of external software solutions. Therefore, it is reasonable to include into solution a compliance passport, a type of passport similar to the requirement passport that is delivered together with the software. This compliance passport fixes and indicates compulsory requirements of the external systems and parameters eventually conflicting with the software. The first application of smart technologies is related to the field of delivering and installing (distributing) of software. Information system equipped with smart technology is able to analyse the environment which it is put into from viewpoint of standard and specific parameters. As standard parameters are supposed to be for example the operational system, the data base management system, browsers etc. used on the concrete server or workstation.

Specific parameters are checked for evidence of previous (possibly damaged) software versions as well as evidence of other specific solutions on the workstation. A typical example of a dangerous software - “neighbour” are antivirus solutions which can classify smart technology software as a virus and even block it up.

Similarly, as for installation of a new version SW users should have a reverse link to SW developer. Due to this reverse link developers can receive complete information on performance, including failure reports and statistics on activities that would allow developers to improve SW quality. As a rule including of above mentioned features into SW requires components that are functioning throughout the whole life cycle of software and are considered as the extension of core functionality that sometimes client remains unaware.

3.2 Installation of Software Version

The solution and installation of the new software version highly depends on architectural design.

Traditionally, web applications are the most uncomplicated in terms of installation, for their operation they use standardized Internet browsers for data illustration. In fact, at least three problems refer to updating process of web applications. They are as follows:

1) **Partial conformity of browsers**. Development of Internet and other browsers is a natural process that cannot be influenced by system developers. Therefore, in practice it is impossible to ensure identical functioning (at least display of information) of all available browsers.

2) **Decentralized data storing**. Companies and public institutions usually do not store their data in centralized system but on individual servers; therefore software and its modules have to be developed and distributed on many mutually independent servers.

3) **Different settings on local workstations**. Personalization as a solution for increased user comfort becomes more and more popular, however conflicts

with the data centralization idea. Additional difficulties arise from different regional settings, that are necessary in exploiting different systems, e.g. decimal separator, date format).

Furthermore, the paper deals with client-server software [7, 8] version distribution due to importance and sophistication of the proposed solutions over centralized web applications.

The installation of the latest software version is done automatically through the Internet and includes the following steps:

1) Installation of the latest version on the server

The latest software version together with the parameter file containing numbers of the latest version of system components and parameters necessary for system operation are placed on centralized server.

In order to enhance version downloading the system is dividend into components, which are relatively independent parts of the system and can be updated independently. For each system component the number of the latest version as well as implementation parameters are recorded in the parameter file.

If the latest version of one of the components is dependent on the latest version of other component then numbers of for both components are increased in the parameter file.

2) Identification of parameters of the latest version

At the moment when a user from the local workstations connects to the system, a built-in mechanism turns to central database and downloads the parameter file of the latest software version.

3) Comparison of software versions

The numbers of the latest version of system components are compared with the numbers of component versions on local work station. For those components whose workstation numbers are smaller than the number on central server, downloading of the latest versions should proceed.

4) Downloading and installing new components

The system downloads the necessary batch of the latest software versions and installs them onto local workstations. During the installation, conformity of the operating system with the parameters indicated in file is controlled. The successfully finished installation has the fixed parameter file on the local workstation by equalizing the version of installed components with the number of installed version. If for some reason downloading or installing of the latest version has failed, version numbers of components in parameter file are not modified, respectively updating of components will be repeated next time when system is turned on.

3.3 Data Migration

Software modifications are often caused by changes in database structure, settings, exchange formats etc.

Data migration as an integrated part of automated version updating mechanism can be analyzed from two aspects: first, matching data structures to the new software version, and second, periodical data synchronization among local workstations (sometimes might work off-line) and central data pool.

Adaptation of data structures to the new software version is ensured by special scripts that are distributed together with the latest software version and activated immediately after software installation. Frequently, complementing data structures is related to complementing historical data with new data, e.g. default, or data migration (structural changes). Though this is relatively simple task, data synchronization is a much more complicated task.

Data synchronization represents a much more complicated task. Data synchronization comprises the main problem when software has client-server architecture and work is ensured in both off-line (on workstations) and online (connecting to the central server). Within data synchronization a threefold task should be solved: first, problem of unambiguous object identification, second, rollback functionality, third, automated synchronization of conflicting and inconsistent data.

One of the most commonly used solutions is extraction of identifier segment, where objects are identified by belonging to a certain instance. If solution is run online, all objects are stored centralized. If, for some reasons running of system online is not available or not used, the system automatically switches to working offline and ensures data storing on local data base instance. After online connection is established, software automatically performs data synchronization by referring to object identifiers and object creation/modification timestamp. During synchronization, instance of data creation (local working station) is identified by using object interval. Then values stored in central database are compared with the actual values taken form local databases and if necessary outdated values in central database are substituted with the actual values.

The specific data migration is needed in those cases when personalization for each software user is provided. If personalization is kept on each individual workstation, then installation of the latest software version initiates migration of the personalization parameters to the new version. If personalization is centralized, the migration should be done on server level. A very characteristic example of personalization is user and rights administration, which is an integral part of any system.

In case of smart technologies migration of the personalized data (user and rights control) is automatically done by internal component, thus relieving user form working with incomprehensible scripts.

3.4 Self-Testing

A very important feature of smart technologies is self-testing. Like living beings who can control themselves and are able to understand the limits of their possibilities, a smart software must be able to check itself before it is run [9,10]. In the hardware industry it is a usual feature that the equipment tests itself as soon as it is switched on. In the case of smart technology software the self-testing means the ability of software to run predefined set of test cases in the production's environment automaticly to check its status.

Functionality of self-testing contains two basic components:

1) Test cases of critical functionality (data component)
2) Built-in mechanism (software component) providing automatic running of test cases

The critical functionality is a set of system functions which are substantial for using of the system. It is impossible to use the system valuably if these features do not function. The critical functionality covers all basic functional requirements of a system – for instance, calculation algorithms, workflows etc. - but does not include non-functional requirements like performance, navigation, and others. To implement self-testing functionality the developers should prepare tests that are appropriate to the selected testing parameter [11] (e.g. full set of test cases [12,13,14]) covering all critical system's functions and incorporating them into the software and the database/ file system.

In addition to that an automatic test running and comparing of results with benchmark values should be implemented [15]. The most important feature of self-testing is an ability to test the software in the productive environment using real data in the read-only mode without changing data entries in production database.

The key feature that differs self-testing from conventional testing [16] is running of self-testing in the productive environment, including an access to the real database of system. The self-testing should be executable in nearly every moment of the system's functioning without disturbing users.

Moreover, each self-testing session includes regressive testing accordingly to the accrued test data. Along with changes in the system a test set and benchmark values are modified/improved accordingly.

System self-testing differs significantly from the traditional testing processes passed during the development phase and performed by the group of independent experts. Self-testing involves developer, thus ensuring that developer's work is not only the written source code but also the result of source code and module tests that ensure self-testing function. Regarding the accrued tests the developer has to prove conformity of his/her work results with the requirements. The need for this procedure appears especially when problems occurred in production environment can not be transferred to the test environment.

Role of self-testing in software development life cycle differs from the role of testing in traditional life-cycle, as for example in model „V". The self-testing in software development life cycle has double application - first, it is a tool for developer to check system operation before testing by the group of independent experts. Second, it indicates readiness of the system tested in production environment.

Though self-testing suggests an increased possibility that the delivered software is a high quality and reliable, it should be emphasized again that implementation of self-testing function requires additional development efforts as well as designing of specific architecture.

3.5 Back-up Administration

Automated version updating modifies application set, environmental settings as well as database structure and even data. Therefore back-up administration should be integral part of the automated version updating mechanism.

The above mentioned system components (application, database, environmental settings) should be provided with two following groups of functions:

1) Back-up generation
2) Roll-back functionality to ensure recovery of prior state in case of failure

Back-up generation is a relatively simple task – before downloading the latest version into special directories software files are copied and databases back-up copy is saved. Also a file with environmental settings is filled (conform to a software requirements passport described in previous sections).

Roll-back functionality is a much more sophisticated task, since, first, due to limited hardware ability duration of historical records is finite, second, recovery of environment to the prior state is not always possible just by modifying standard settings.

Important part of roll-back functionality is detection of failures, respectively, set of unambiguous indications that signal failure in installation of the latest software version and necessity to recover to the prior state from back-up files. Moreover, due to the limited ability of keeping historical records it is absolutely crucial that generated and available back-up files are created from feasible software version not from the previously failed installation efforts! The analysis of failures, especially, the set of indications signaling capability or incapability of the software is a subject of separate research paper. In most simplistic solutions all components are checked and main application window is opened without getting to the error handling routine.

In our proposed solution the roll-back functionality ensures repeated downloading and installing of the last successfully installed version including reconstruction of the appropriate database structure. Unfortunately, in this case all the data accrued after the last successful installation are endangered; respectively they are saved in modernized data structure though by incomplete software version.

4. Use Cases

The above described principles of smart technologies are implemented and approbated in software development and implementation projects [8,18,19]. However, in neither of software products smart technologies were implemented to the full extent. These software products support only several features; furthermore, functionality of most of these features is narrowed.

Henceforth, the paper deals with the results of particular project, referring to automated version updating.

The task of the project was to develop software that would ensure structured gathering of financial information on several hierarchal levels. The system was supposed to run in 600 public offices located throughout the territory of Latvia ensuring regular gathering of information by different time periods. It was required that the system should be able to run not only in offices provided with Internet connection but also in offices with irregular and instable Internet connection or even in offices without Internet connection. The requirements suggested remote software installation and very limited financial and human resources were allocated for maintenance (user support) services. Furthermore, the specific requirements dictated that in certain periods most users will use the system simultaneously.

An application that provided all required functions has already been developed; therefore project task was to adapt application according to the requirements of automated version updating. Additional human resources for the system adaptation accounted for c.a. 10% of the initial system developing resources. The most part of these 10% were invested into extensive testing and code review. In order to ensure reliability of the developed mechanisms, software testing with different infrastructure configurations was performed on virtual machines.

The achievements and results of this work have been used successfully in all local governments and many public institutions on different levels of hierarchy in Latvia since 2005. Automated version updating has significantly improved system maintenance and reduced the need for user consulting resources. We believe that the idea has proven to the successful and we have continued implementing it in several other projects.

5. Indications for further research:

1) Implementation of smart technology principle in software takes fewer resources than full-range configuration support. At the same time, smart technology places fewer constraints on the acceptable means of expression.
2) Smart technologies allow reducing the efforts for software testing and setting up, thus increasing the client service level significantly.

3) Smart technologies assist to provide software performance in a changing environment and environment containing heterogeneous platforms and infrastructure. Nevertheless mechanisms of smart technologies need regular adaptation to the environment changes, especially in case of standardized software. It is very important to provide in-depht reporting mechanism to inform the developers about indicated problems in time.

4) Adding smart technologies to the software after the development is useful though requires more resources than including smart technology already in the software architecture design phase.

5) Although the clients approve opportunities provided by the smart technologies, usually, they are not willing to provide additional financial means to ensure them. The smart technologies are certainly costly – effective and should pay-off in long-term business projects and long-term cooperation with client when the number of users exceeds 10 or if workstations are configured differently. Opportunities provided by the smart technologies pay-off and enhance software distribution and user support even if a company has only two geographically separate subdivisions.

Acknowledgements

The research is supported by the European Regional Development Fund (ERDF).

References

[1] Bicevska Z., Bicevskis J.: Smart Technologies in Software Life Cycle. In: Münch J., Abrahamsson P. (eds.): Product-Focused Software Process Improvement. Lecture Notes in Computer Science, Vol. 4589. Springer-Verlag, Berlin Heidelberg (2007): 262-272

[2] Roger S.Pressman, Software Engineering. A Practitioner's Approach. Sixth Edition. McGrawHill. 2005

[3] Roger Laddaga, Paul Robertson *Self Adaptive Software: A Position Paper*, International workshop on Self Adaptive Software Properties in Complex Information Systems, 2004, Bertinoro, Italy

[4] Qianxiang Wang Towards a Rule Model for Self-adaptive Software ACM SIGSOFT Software Engineering Notes, January 2005, Vol. 30, N. 1

[5] Rauhvargers K. and Bicevskis J., Towards a semantic execution environment testing model (this volume)

[6] Beydeda, S.: Research in Testing COTS Components - Built-in Testing Approaches. In: ACS/IEEE 2005 International Conference on Computer Systems and Applications. Bonn, Germany (2005)

[7] Andzans A., Mikelsons J.,Medvedis I. And others. *ICT in Latvian Educational System - LIIS* Approach, Proceedings of The 3rd International Conference on Education and Information Systems: Technologies and Applications, July 14 - 17, 2005, Orlando, Florida, USA

[8] Latvian Education Informatization System – LIIS [on-line]. Available on the internet: http:/www.liis.lv

[9] Sami Beydeda: Self-Metamorphic-Testing Components. COMPSAC (2) 2006: 265-272

[10] Beydeda, S.: Research in Testing COTS Components - Built-in Testing Approaches. In: ACS/IEEE 2005 International Conference on Computer Systems and Applications. Bonn, Germany (2005)

[11] Boris Beizer. Black-Box Testing Techniques for Functional Testing of Software and Systems. John Wiley & Sons, Inc, USA, 1995

[12] Bicevskis J., Borzovs J., Straujums U., Zarins A., Miller E.F. SMOTL - A System to Construct Samples for Data processing Program Debugging. IEEE Trans. Software Engineering, 1979, SE-5, No.1

[13] Bicevskis J., Bicevska Z., Borzovs J. Regression Testing of Software System Specifications and Computer Programs. Conf. Proc. Quality Week, San Francisco, 1995, Software Reserch Institute.

[14] Bicevskis J. The Effictiveness of Testing Models. Proc. of 3d Intern. Baltic Workshop “Databases and Information Systems”, Rīga, 1998

[15] Hans Buwalda Essentials of Testing and Test Automation. Proc. of 15th Quality Week 2002, San Francisko, 2002

[16] Surya Kumar Role of Test Tools in Product Testing and Automation. Proc. of 6th ICSTEST Conf, Dusseldorf, 2005

[17] A.Auzins, J.Barzdins, J.Bicevskis, K.Cerans, A.Kalnins. Automatic construction of test sets: theoretical approach. Lecture Notes in Computer Science. Vol. 502, Springer - Verlag, 1991.

[18] [on-line]. Available on the internet: http://www.numuri.lv

[19] [on-line]. Available on the internet: http://www.liaa.gov.lv