Baltic J. Modern Computing, Vol. 8 (2020), No. 2, 241-258
https://doi.org/10.22364/bjmc.2020.8.2.03

Experience in Automated Functional and Load Testing in the Life Cycle of the Mission-critical System

Boris POZIN[1] , Ilya GALAKHOV[2]

[1] EC-leasing, 125 Varshavskoye shosse, Moscow, 117405, Russia, National Research University Higher School of Economics
[2] EC-leasing, 125 Varshavskoye shosse, Moscow, 117405, Russia

bpozin@ec-leasing.ru, igalakhov@ec-leasing.ru

Abstract. The article presents the results of an analysis of the experience of automated testing of mission-critical systems during operation, maintenance and development over 12 years. It is shown that in these processes, compared with the development process, significant changes are taking place in the goals, criteria, methods and technology of conducting functional and especially load testing of an industrially operated and accompanied automated system. The main goal of the work on testing the release is to plan and control the integrity level of the release of the system, as the main criterion for the quality and continuity of the business using an operated automated system. It is shown how automated system testing in these processes is used to assess the probability of fulfilling business production regulations using the system, that is, business continuity, as well as to assess the ability of achieving the necessary operational characteristics of the system when changing data processing technology, load, or when scaling the system. The experience of planning various types of load testing using the models given in (Pozin and Galakhov, 2011) is considered. The technology and a set of tools for automating functional and load testing developed by the authors are described.

Keywords: Business continuity, System integrity level, System/software release, integrity level requirements

1. Introduction

Among automated systems (AS), a rather significant part is made up of the so-called mission-critical systems. The duration of their life cycle (operation, maintenance and development) is 10-25 years. As a rule, this is a system with the owner (company or government), who is not a developer of the AS or its software (SW), but is responsible for the organization, maintenance and development of the AS and the compliance of the operating characteristics and quality of the AS with the requirements of business or government (depending on purpose AS). Systems of this class perform, first of all, the functions of a “back office”, that is, accounting for financial and material flows,

analytical systems of the state or corporate level. More and more systems of this class carry out centralized data processing. In such systems, in view of the need to ensure state or commercial secrets of the data processed and accumulated in the system, there is an AS information security perimeter.

Such systems are in most cases custom-made. As the owner of the system perceive automation capabilities, and the developers of the AS - the subject area, the AS is constantly being improved on the basis of the business needs formulated by the system owner as a customer. The effectiveness of the system is determined by how much it allows to improve the quality of the business, that is, to improve the ability to meet the needs of the owner of the system and its customers, reduce the complexity, the duration of solving the business problems of the organization-owner, and also to increase labor productivity when performing business processes of the company within the specified time constraints.

It is unacceptable for the AS to negatively affect the business continuity of the owner, while the level of business continuity must be formulated. For example, "when the system is in 7 * 24 mode, the recovery time after a probable failure should not exceed T minutes". T is selected taking into account the prevention of material or reputational losses of the owner and should be regulated in the internal documents of the owner company.

The phased perceiving by developers of the needs of the business and understanding by the owners of the system - the available opportunities leads to the system (including application and system software) is developing in stages by releases that are generated and put into operation on a regular basis (once a month, quarter, etc.) in accordance with some maintenance and development plan (usually annual).

The new release is distinguished by the fact that, in relation to the previous one,

- the detected errors are corrected (in volume - according to the agreed plan);
- software subsystems with new functionality were included into the system, and / or the functionality of existing subsystems was changed;
- the subsystems are reworked or the operating environment of their execution and / or the parameters affecting the performance of the functions by the subsystems are configured so that the operating characteristics of the system are within the specified limits.

The rate of change is quite high (several hundred or thousands per year), therefore, when putting into operation each release of the system, it is necessary to maintain its integrity (ISO/IEC 14764, 2006), that is, the implementation of the planned functions and ensuring the achievement of the specified operational characteristics (performance, reliability, etc.), which are necessary to carry out production business regulations of the organization. Thus, the provision and control of sustaining the integrity level of the system is the goal at all stages of the life cycle of the system: during maintenance, development and modernization. Integrity level is “a denotation of a range of values of a property of an item necessary to maintain system risks within tolerable limits. For items that perform mitigating functions, the property is the reliability with which the item must perform the mitigating function. For items whose failure can lead to a threat, the property is the limit on the frequency of that failure” (ISO/IEC 15026-3, 2011).

In essence, this means that each release of the system (including the release of system software) must be monitored to maintain the integrity level. Violation of the integrity leads to the implementation of systemic risks, that is, to disruption of business continuity, and therefore to losses for the business - both material and reputational. Different types of risks can have different consequences for a business.

Several releases of the system are put into operation in a year at a pace in which the staff is able to master and apply the changes in practice (in business). Each release should be characterized by an appropriate level of integrity, at which the main planned functions are implemented in it, and the use of the system under the planned load does not lead to violations of business continuity (established business regulations) when performing the most critical business functions.

Such a statement of the problem poses some additional requirements for the technologies and methods for testing software in the system, in contrast to those published in the scientific and technical literature (Zhen and Ahmed, 2015; Klemm, 2013; Molyneaux, 2014).

The literature on testing technology (Zhen and Ahmed, 2015; Blokdyk, 2018; Dürr, 2010; Glavich and Farrell, 2010; Klemm, 2013; Molyneaux, 2014; Subraya, 2006) practically does not cover issues related to how to organize testing processes during the operation of the system and its development by releases, what are the goals and objectives of testing in this case, what types of testing are appropriate to use. A number of works (Lysunets and Pozin, 2013; Gotsutsov, 2017) discuss or solve similar problems. Unfortunately, the general statement of the problems of testing during the operation and development of custom-made systems has not yet been formulated. However, it is highly desirable to develop an approach, methods and technology for solving this problem. This work is one of the first steps in this direction.

2. Statement of the problem of supervision the system integrity level

In our opinion, the approach developed in the international standard (ISO/IEC 15026-3, 2011), which considers a model based on a systematic analysis of system integrity levels in risk analysis, seems to be the most suitable for setting such a task. It is this approach that reflects the state of the system during operation and development, as well as the owner's point of view on the system. With this approach, it is possible to compensate for factors that potentially violate the integrity of the system: uneven commissioning of programs that implement new functional requirements for the system, the difficulty of confirming non-functional requirements that ensure business continuity, etc.

Compensation of the influence of these factors can be achieved by

- testing planning, in which it is possible to take into account the potential change from release to release of acceptable risk levels that occurs due to the quality of the changing components of the application software of the system or the system as a whole being put into operation as part of the release;
- implementing the plan in the course of preparing the release for putting into operation.

Almost immediately after the system starts to operate, the task of supervision the level of integrity of the system when making changes to it arises. As a rule, in the process of developing a system, many integrity control tasks are not yet recognized, primarily because there is not enough information and experience in using the future system in the owner's business processes to formulate and solve this problem at the development stage.

Typically, developers by the functional integrity of systems understand the degree of completeness of the implementation of the planned changes in functionality. The most often considered as non-functional characteristics of systems are their performance (the

number of realized requests to the system - on average, or by the types of processed requests with some background load), the speed of query execution under the background load (on average or by types of requests), etc. In practice, this approach is far from always acceptable. An automated system is just a tool of the owner company to improve its business processes: to improve the quality of customer service or to reduce costs. Typically, the company establishes regulations the execution of certain business objectives, taking into account the real possibilities of the staff and the level of automation of the company and its contractors. These regulations, among other things, establish acceptable time intervals for solving business problems. To solve a business problem of a certain type, it may be necessary to perform several queries to the system, including more than once. Failure to complete sets of business operations, including those supported by automation tools, within acceptable time intervals may mean loss of income and / or reputation for the business.

From the point of view of the owner of the system, supervision the integrity level means not only controlling the degree of completeness of the planned changes in functionality, but also the ability of the system to provide unconditional fulfillment of regulations at a given load: solving complex tasks for a business at time intervals established by regulations. Under the load refers to the number of requests for solving business problems per unit of time.

Violation of regulations may be critical for business (Lysunets and Pozin, 2013) or have weaker levels of criticality. The ranges of changes in criticality levels are set in each company taking into account its business interests based on an analysis of regulations. Assessment of the integrity level should include all the described components of the implementation of the regulations: on the functionality of the system and on non-functional characteristics, taking into account the criticality of the company's business processes.

Thus, in essence, the following model of the integrity level control process is considered:

Let the set {M} of automated business processes of a company be given, each of which over a period of time T is performed with probability p_i (t), i={1,M}. Let each business process be implemented by one or several chains of j subsystems, j={1,J};

In this regulation the fulfillment of a business process is set so that

$$\forall j \ \max_j t_j \ \leq \ \tau_i, \tag{1}$$

where

tj – the execution time of the chain j,

τ_i – is the regulatory restriction on the execution time of business process i.

Each chain j consists of software, which may contain a residual critical error inherent in this software in the current release. This residual error may occur when running the release with probability qj. The manifestation of residual error characterizes certain level of integrity for solving the problem of a particular business process, rather, execution of the corresponding chain j. The probability of the realization of risk p_r in this case is

$$p_{ri=} \ p_i * q_j \tag{2}$$

In this case, the risk level is characterized qualitatively (Lysunets and Pozin, 2013) by the value of φ_i so that for the entire release the probability of occurrence of risk is $p_R = \bigcup_i p_{ri} * \varphi_i$.

If φ_i = 1 at a critical risk level and 0 in other cases, then for critical chains:

$$p_R = \sum p_{ri}$$

The task of supervision the integrity level is reduced to the procedures:

- determining the expected risk levels for the release of the system;
- describing the functionality of the release, corresponding to the occurrence of risk in planning the monitoring of the integrity level;
- carrying out comprehensive functional testing of the release to identify residual system-plan errors (for example, those associated with incompatibility between the data or the boundaries of the domains of the variables of new components with unchanged components).;
- carrying out regression load testing to detect cases of residual errors;
- carrying out regression load testing to detect cases of potential violation of regulations (in terms of performance or efficiency of solving problems of chain j).

The absence of residual errors or violations of non-functional requirements for the release of the system during comprehensive testing of the release allows us to conclude that the release is verified. The release is put into operation and its validation is carried out within 1-2 weeks (depending on the technology applied by the owner). According to the available experience, the release works stably after the control period during which no more than 0-4 critical errors appear (and are eliminated) (Lysunets and Pozin, 2013; Gotsutsov, 2017).

3. Release test planning

The need to control the integrity level using the described approach leads in practice to the organization of testing processes as iterative, regression, ensuring the achievement of the assigned tasks by the planned deadline for putting into operation a new release of the system based on some typical comprehensive testing plan.

The planning of testing is based on achieving the goals of creating a release, taking into account the general technology of work to integrate the release on time and ensuring acceptable risks, which are manifested through the control of the integrity level using methods of comprehensive testing. The model of risk levels, containing a qualitative assessment of the risk level of each chain, is quite stable as a whole for a system whose structure does not fundamentally change from release to release. Only for several (one or several) chains of a new release the level of risk can be clarified. For example, if the occurrence of a release is associated with a change in regulatory documents, then the priority of certain business tasks may change, as a result of which the risk level of the chains that automate these tasks will also change. The model of risk levels is agreed or approved by the system owner or his authorized person.

Thus, the main purpose of testing each new release is to determine the changed chains, control their functional integrity from the position of the system, and then check how the integrity violation (manifestation of residual error) can affect the integrity of the entire system when the system operates under load. Achieving the goal is carried out according to a single technology, understood and approved by the owner of the system, since it is the representatives of the owner who make the final decision on whether the release of the system meets the specified level of quality. If there are resource or time restrictions on the examination of the release by the planned date, the testing plan is refined taking into account the priority of the chains under modification by integrity levels. The priority ones to exam in the current release are those chains that have higher risks.

3.1. Planning for comprehensive functional testing of the release

Planning for comprehensive functional testing is used to minimize the cost of resources (primarily the time and laboriousness of test development) for this type of testing. In accordance with the integrity level scheme and release plan, it is enough to simply plan control over the coverage of implemented functions by tests, on the basis of which to develop criteria for the completeness of comprehensive functional testing of the release. According to the release development plan, it is determined which chains will be affected by the changes that will help plan the composition of complex tests to verify the functional integrity of the release.

Planning for comprehensive functional testing of the release aims to control three main aspects:

- degree of implementation of the planned new features included in this release in accordance with the release plan;
- implementation of planned improvements to previously implemented functions that were not correctly implemented in previous releases;
- the impact of newly emerging and existing information links on the correct operation of the release, primarily on the dependability of functions that have not been modified.

Since the control procedure is mandatory and is carried out regularly for each release, it is fundamentally possible to carry out control according to the checklists of the items in the release plan, and in some cases, this control can be automated for certain aspects (see below). The checklists should essentially reflect the degree of implementation of the release plan and the impact of the implementation of the plan on certain types of risks and integrity levels. The results of this control are compared with the model of risk levels for all chains that have undergone changes. Additionally, for the software as a whole, tests are run that verify the third of the listed aspects. In essence, this type of verification reduces to checking the input variables of each of the most important chains of software components in the entire range of their changes. This will make it possible to control the compatibility of new software components concerning information relationships, as well as to monitor the absence of the influence of new information links on previously developed and unchanged components. It is most efficient to automate this procedure. The possibility of its automation is described below.

3.2. Planning for comprehensive load testing of the release

An experimental assessment of achievable levels of integrity of non-functional characteristics achieved in the release is carried out by the method of load testing. Essentially, at this stage, load testing is carried out as verification, since it assesses the extent to which the goals of creating the release are achieved, and how much the requirements for it are met in the release.

Since the resource consumption, laboriousness and, as a result, the cost of load testing is extremely high, it must be planned based on the current testing goals and the acceptable costs of achieving them.

The key aspects when planning an experiment to load testing of the release are:

- the statement of the problem, which defines the goals of the experiment, must be related to the requirements for the system;
- a clear definition of the boundaries of the test object;

- a scenario of the functioning of the system during the semi-natural experiment, which reflects the features of the intensive use of business processes of the organization-owner;
- conditions for the maximum load entering the system during the experiment, based on the features of the scenario;
- a set of necessary characteristics and indicators, on the basis of which the test results are determined.

The starting point for planning is a scheme of integrity levels. First of all, it is estimated whether the values of the integrity levels of the release in question are preserved or changed compared to the previous one. The most critical business process regulations should be identified, the potential for violation of which are critical. Control of compliance with these regulations (and to what extent), becomes the target of load testing. The composition of critical and important business processes that implement these regulations is being formed.

For further planning of load testing, the approach described in (Pozin and Galakhov, 2011) is used. Together with the owner of the system, the goal of load testing of the release is concretized, since in addition to monitoring the integrity of non-functional requirements, it can also have a meaningful wording. For example, to reduce the time for generating final reports in such a direction by 40% compared with the previous release.

Based on the critical and related business processes selected for the goal, the composition of which is agreed with the owner of the system, a requirements model is created (Pozin and Galakhov, 2011), with which a load model is also formed. The load model fully describes the mix of user queries to the system on which the achievement of the goal of load testing is evaluated. The definition of the boundaries of the test object is formalized in the model of the system. The system model is usually stable for multiple releases. It can be modified when the composition of the hardware or system software changes in the release of the system, or if it is necessary to analyze not the entire system, but only its parts. The measurement model practically does not change from release to release, since system software changes extremely rarely, and the way to obtain primary information about the characteristics of a computing process is entirely determined by its structure. In addition, it is important that the results of load testing of different releases are comparable, including the units of measurement and the accuracy of the primary data.

Thus, when planning load testing, it is necessary to follow the path of integrity control from the integrity level scheme to critical business processes, then to the regulations, from them to the load (mix) and to the integrity control according to the criteria for fulfilling the regulations. Note that this method of planning load testing ensures proof of the adequacy of the results of load testing.

These aspects are agreed with the responsible persons of the organization-owner of the system in the form of a document - even if the actual load testing is carried out by a division of this organization, the affiliated organization-tester, or even more so an outsourcer. The listed works should be performed before the start of the load experiment.

4. Tools for comprehensive testing during operation, maintenance and modernization

Comprehensive testing of releases is essentially black box testing with the condition that a set of requirements (functional or non-functional) for the system or its software is known.

A set of functional requirements is concretized by a system use-case (or by a set of such system use-cases) that implements each requirement. For each system use-case, the conditions for its execution and the composition of the input and output data are formulated. To verify the compliance of the system with the functional requirements for it, test requirements are developed containing methods for checking each functional requirement. The functional testing technique provides for the implementation of the following steps: test planning, preparation of test data, test run, evaluation of the results of the run, assessment of the degree of completion of testing.

To verify each test requirement, it is necessary to plan the corresponding test cases, expected test results and criteria for the completeness of testing and its completion, as well as the composition of system-wide data on which to test the system: the necessary databases and their tables filled with data, reference data used in an automated business process. Thus, the plan for comprehensive functional testing of a system's release is a sequence of checks of new or refined functional requirements for a release plus a previously existing package of regression checks of requirements that did not change with respect to the previous system release. An additional condition may be to check some or all of the new requirements on an extended range of input data values in order to identify the degree of influence of changes in the values of such data on the correctness of previously unchanged parts of the previous release of AS software. Note that the volume of new functionality (new system use-cases) in the release in relation to the functionality of the entire system is usually not more than 5%.

The set of non-functional requirements is the above-described requirements model and the scenario realizing the goal of the load experiment, according to which the load model is built. The load model is essentially a time chart taking into account the technology for implementing each requirement in the form of a set of system use-cases with a characteristic of the intensity and duty cycle of the initiation of each system use-case.

Such a look at the release of the system as an object of testing provides a key to solving the problem of generating test data for complex verification testing of the release. With this approach, all the data necessary for the preparation of test sets is known, and it is necessary to find a method by which the process of generating such data can be automated.

Essentially, two tasks must be solved: determining the composition of the test data for each test requirement (test), and choosing the method for feeding test data to the system input during each test run. The solution to the second task depends on the architecture of the system being tested.

System architecture, as a test object, in a centralized data processing in terms of input signal types, can be divided into (primarily) user-driven and message-driven. User-driven systems include systems built on a client-server architecture in which end-user workstations are directly connected to application servers of the processing center. From these workstations, users form requests to the system using the technology described in system use-cases. In message-driven systems, messages are used as a unit of exchanging

information between parts of systems, that is, the end user does not interact directly with the processing center. The processing center receives messages from the components of the system and also responds by sending messages. Depending on the purpose, messages can be divided into types. Messages containing electronic documents for processing, in turn, can be divided into types in accordance with their formats and priority. In addition, messages can be:

- single (containing one electronic document);
- packages (containing a set of electronic documents).

Depending on the sender, each message can be encrypted and signed with one or more electronic digital signatures.

From the test point of view, for each test, in addition to input data (messages), it is necessary to have output reference standard data. When processing a message, the internal databases of the system are used. Such databases may contain metadata, that is, from such databases information about the objects being processed is read (for example, the list of airports permitted for this type of aircraft is in air traffic control systems). Databases may also contain records with the results of processing input data. If changes to such records are significant for assessing the quality of implementation of a functional requirement, then, based on the results of a test run, it may be necessary to provide access to the corresponding record to compare it with the reference standard. Upon completion of each test run, before starting a new test, the initial values of the variables of the changed record must be restored to ensure the testing is correct.

Thus, a test to verify the test requirement r can be represented as:

$$T_r=\{D_{ir}, D_{oer}, D_{dber}\}, \quad (3)$$

where

T_r – test requirement to verify;

D_{ir} – set of input values of test data for verifying the requirement T_r;

D_{oer} – set of output values reference standards for verifying the requirement T_r;

D_{dber} – set of internal database data reference standards for checking the requirement T_r.

For one verification test plan for the release of the AS, it is necessary to prepare a very large amount of test data:

- the amount of input test data in various forms (for example, XML messages) - several hundred;
- the amount of data for comparisons with reference standards -several hundred or more;
- the number of reference standards - several hundred.

With limited time resources for complex testing of the system release, the complexity of manual preparation of such data is very high, as well as the complexity of analyzing the results of the test run.

Of course, a certain amount of manual comprehensive testing in conditions of time pressure is inevitable, but the process itself makes it possible to automate the generation of test data based on existing test requirements and information about the data contained in the system (release) under test. This paper presents the main directions of automation of the processes of generating test data and comparing test data and reference standards, as well as solving similar problems in regression testing, where the amount of relevant data increases by at least an order of magnitude.

Depending on the type of system, either emulation of user actions or sending input messages is used for testing. Accordingly, the tester must analyze the response of the

system either in the form of responding to user requests or in the form of output messages. The essence of the analysis, as a rule, is reduced to comparing the output signals with the reference standards. For example, output messages are compared with pre-prepared reference output messages that are expected to be received in response to specified input messages if the system functions correctly.

During the period of setting up work on testing systems for the owner organizations, such tools were not available on the market, therefore, we developed the necessary tools, went through many years of use in practice, and implemented them in various operating organizations and maintenance organizations.

5. Methods and tools for comprehensive functional testing

Methods of automated regression functional testing. The tasks of complex functional testing can be either verification or validation of the functioning of the system. During verification, the system is checked for compliance with the specified specifications of functional requirements. Validation is intended to verify how much the system corresponds to real business processes, that is, how much its functioning corresponds to that expected by the owner of the system (end user). To verify message-driven systems, a method for comparison reference standard messages and database table states has been developed. For system validation, the gross testing method is used (see below).

In the process of verification testing, the tester must compare the processed value of the set of variables and fields of database records with the reference values. For each test requirement, it is necessary to compare several hundreds of such pairs of values, including values in code form (for example, binary). The total power of the comparisons set is bottom estimated as

$$| \mathrm{T\,r} |\mathrm{r} = \sum_{\mathrm{r}} | \mathrm{Doer} + \mathrm{Ddber} |.$$

In view of the significant amount of analytical work and the time limits for conducting comprehensive functional testing, it turned out to be expedient to automate the comparison processes provided that the tests are described in the format (3).

The method of comparison of reference standard messages and states of database tables is intended for testing systems that interact through sending and receiving messages. Typically, in mission-critical systems, message delivery is carried out by guaranteed message delivery systems, such as, for example, Websphere MQ Series. In such systems, queues of incoming and outgoing messages are configured, which, in fact, are the input and output of the system under test. To prepare the test, it is necessary to form two types of reference standard messages: input and response. When the test runs, the tester places the reference standard input message in the incoming message queue. After the system has processed this message, it is necessary to receive a response message from the outgoing message queue. The received response message is compared with the reference standard response message. If they coincide, we can say that the tested system worked out as expected. Otherwise, a defect should be assumed.

When forming the regression tests, the response messages received as a result of processing the test input messages using the tested version of the system are considered as reference standard response messages. Thus, it is expected that each new version of the system, upon receipt of the same test input messages, will, as a result of their

processing, produce response messages that match the previously recorded reference standard messages.

All stages of this testing method, from the input of test data to the comparison of the results of the received response messages with the standard ones, as well as bringing the internal tables of the system after running the test to the initial state, are currently automated using the developed Automated Testing System (SAT) operated by more than 10 years.

It should be noted that the internal state of the system can affect the expected results. Therefore, to perform testing, a reference standard database is required that is consistent with the test data. For each test run, such a base should be brought back to its original reference state to ensure the comparability of test results. Correspondence of the obtained result to the standard does not guarantee that when it is formed, the system has switched to the state it was supposed to go into. That is, externally, the system may look impeccable, although inside some functions may work out with errors. As a rule, in information systems, with their competent design, it is the database in full that reflects the state of the system. Therefore, in addition to preparing the reference standard input and response messages, the tester prepares a template for changing database tables. At the regression testing during the test execution, the automated testing system automatically registers database state changes and compares it with the reference one (see Figure 1).

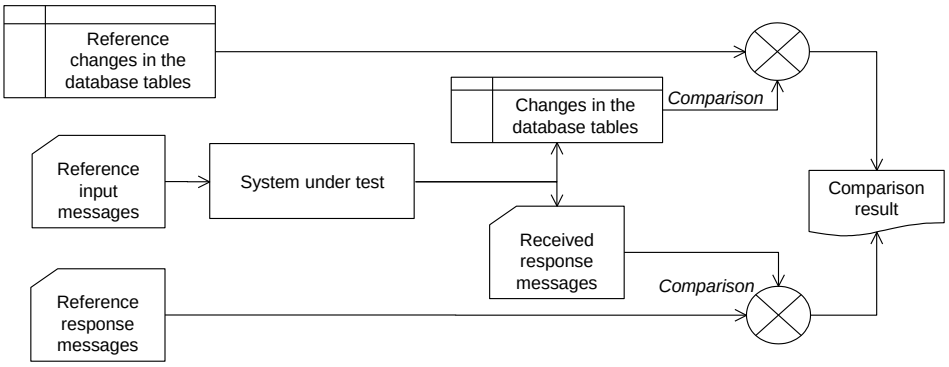

Fig. 1. Testing scheme by comparison of reference standard messages and database table states with got ones

If response messages match with the reference standard messages and changes in the database tables match with the reference ones, then with a high probability the tested functionality in the tested version of the system has not deteriorated. Of course, the method will work only with the reference standard database. The reference standard database guarantees repeatability of tests and comparability of test results.

The method for generating test data for equivalence classes of the domain of definition of basic variables. In regression testing, in a number of cases, primarily when assessing the impact of information connections of new components on the reliability of functions that have not been modified, it turns out to be expedient to form a series of tests that differ in the values of some significant variable. This is a fairly common situation (from 6 to 10% of the total number of regression tests). To generate reference test data in such cases, a specialized tool was developed - a test message generator for

equivalence classes of the base variable (ISO/IEC/IEEE 29119-4, 2015; Gotsutsev, 2017). The testing scheme using the generator is shown in Figure 2.

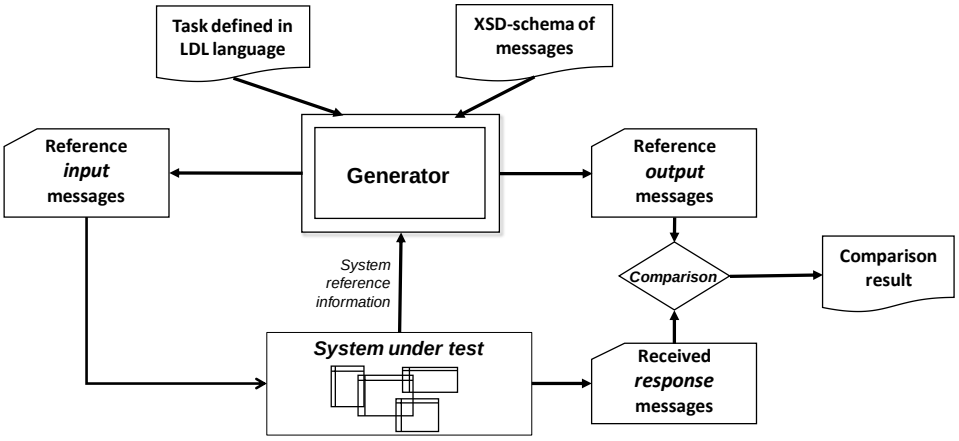

Fig. 2. Testing scheme using test message generator for equivalence classes

Using the LDL language (Pozin et al., 2017), when using a generator, the tester describes the domain for determining the values of the variable that is part of the message from the number included in the message library of the system under test described by the XSD scheme. The generator constructs equivalence classes in accordance with the algorithm described in (Pozin et al., 2017), and generates many test messages and reference standard response messages, which are then used for comparison. Using the generator can significantly reduce both the laboriousness of the preparation of test data, and the laboriousness and duration of the analysis of test results.

The method of gross testing. Some systems archive input and response messages. This makes it possible to use such messages as tests, re-applying them in comprehensive functional testing. Because the described messages have already been processed by the system under test, such testing occupies an intermediate position between verification and validation. All data in the messages is veritable, but some of the data may be confidential. Because comprehensive functional testing, as a rule, is carried out outside the production information security perimeter, when transferring archived messages to the testing perimeter, confidential data must be depersonalized. This procedure is automated using algorithms similar to those used in the test message generator described above. The degree of completeness of testing by this method is entirely determined by the composition of the archived messages, however, the complex of test messages may contain information links and other relationships between messages that are not always clear to testers. Analysis of test results, as a rule, is carried out "manually", when cases of abnormal passage of test messages are detected. The volume of testing is regulated taking into account the available workforce and time for its implementation. With such testing, the temporal characteristics of message processing and system performance are not evaluated. The testing scheme for this method is shown in Figure 3.

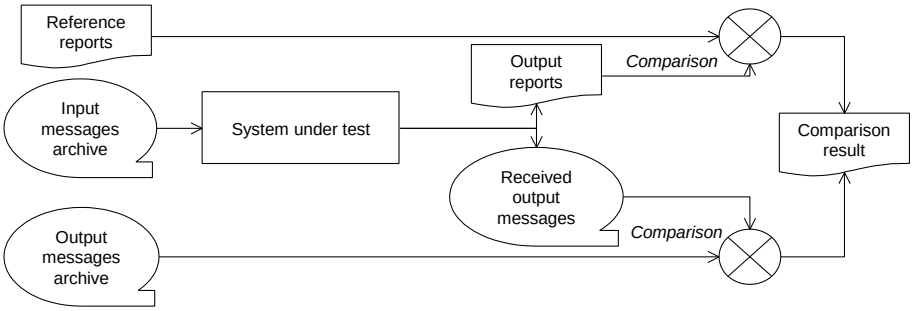

Fig. 3. Testing scheme by gross testing method

An additional way to compare the results of the tested system with the reference standard is to compare the output reports produced at the end of the working day obtained as a result of gross testing with the reference standard reports received at the test bench as a result of the previous gross testing of the version of the software which is in production operation. As already noted, gross testing is performed on depersonalized data. Due to this, unfortunately, reports from the system in operation cannot be used as a reference standard, as they may contain confidential information. However, it can serve as a prototype for the tester, who analyzes the test results.

The use of the described methods of regression functional testing together allows minimizing the risks of errors during operation, since the system is tested both on a fixed reference standard database and on a depersonalized real database. Here are some quantitative characteristics of the volume of automated testing, which is currently being performed using the developed methods and tools.

Table 1 shows the volume of accumulated automated regression functional tests for a mission-critical banking system: the number of tests, the number of input test messages, the number of output (reference) messages.

Table 1. The volume of automated functional regression tests

The message type	The number of tests	The number of input test messages	The number of output (reference) messages
Type 1	131	125	3124
Type 2	194	657	2190
Type 3	117	398	3551
Type 4	472	1741	5946
Type 5	461	774	3864
Type 6	456	2033	4992
Total	1831	5728	23667

These tests are applied to the system on a daily basis during automated regression testing of each release. In addition, the volume of gross testing is several million input messages.

6. Methods and tools for comprehensive load testing

Load testing, in fact, is a semi-natural experiment with the system. At various stages of the AS life cycle, different methods and techniques of automated load testing are used (Zhen and Ahmed, 2015). At the stages of operation, maintenance and modernization of the AS, one of the tasks of load testing of the release is to detect signs of degradation of the AS after making changes to it during the preparation of the release.

The test bench for such experiments should ideally practically repeat the system that is used in production. Only in this case, the experimental results can be adequately extended to the real system. In our practice, a reserved complex of technical equipment was used for the test bench, equivalent in its characteristics to the main one. The experiments were carried out in those days when the stand-by complex was unoccupied.

A full-scale load experiment, as a rule, in this case simulates a full day of the system. This is due to the distribution of load during the day and the features of the business processes of the organization-owner of the AS.

Key aspects when planning an experiment have been reviewed in 3.2.

The authors developed four metamodels (Pozin and Galakhov, 2011), with which, when setting the task of the load experiment, the necessary characteristics, indicators and measured values are selected that adequately characterize the functioning of the tested information system:

1. The requirements metamodel - characterizes the type of system under test, the composition of non-functional requirements (business rules and technical requirements) of the information system under test;
2. The metamodel of the system - describes the structure of the system as a network of queuing systems (including the composition of elements of the "resource" type);
3. The load metamodel - is a description of the number and types of service requirements for the system, the law of the distribution of service requirements in the experiment, the rules for the receipt of service requirements in the system, the entry points of service requirements in the system (logical level);
4. The measurement metamodel determines the composition of the collected characteristics, indicators and quantities, the interface for entering requirements into the system, the method for collecting them and the transformation algorithms, as well as the criteria for evaluating the results.

When planning a new load experiment using the concepts of metamodels, models of requirements, systems, loads and measurements are formed by choosing meta concepts and determining their values based on the properties of the information system under test and the goals of the load experiment. Through the use of metamodels when planning a new load experiment, the completeness and integrity of the formed models are ensured. For different types of load testing and different systems, these models may vary.

The technique of load testing consists of the following steps:

- Determination of testing objectives;
- Development of a “Testing program and methodology”;

- Preparation for testing;
- Load feeding;
- Data collection;
- Interpretation and analysis of results.

At the stage of "Determination of testing objectives" using the requirements metamodel, the rules for formalizing the requirements for system performance are determined and a requirements model is formed. According to the requirements model, the goals of load testing and its scenario are agreed with the customer.

At the "Development of a "Testing program and methodology"" stage, from the requirements model, a description of the scenario, the objectives of the load testing and requirements for the estimated operational characteristics are entered into the document "Testing program and methodology".

At this stage, the load metamodel is used to determine the possible static and dynamic structure and composition of the load flow. Based on the coordination with the customer of the scenarios and the composition of the load flows, a load model for a specific experiment is formed from the load metamodel.

The metamodel of the system allows us to describe the structure of the system in the form of a model of the system as a network of queuing systems. The model of the system contains requirements for a load testing test bench, which are agreed with the customer.

On the basis of the measurement metamodel, the measurement model documents the set of necessary characteristics and calculated indicators measured during the experiment, methods for their preparation and criteria for evaluating the test results.

At the "Preparation for testing" stage, on the basis of the load model, the testing planning tools and the test data generator are configured. Test data is generated automatically in accordance with the quantitative composition of each type of load defined in the load model. The generated test data is placed in the test database.

According to the model of the system, tuning of the tools for checking the test bench and tuning of measurement automation tools regarding data collection points and used collection mechanisms is carried out. The tuning of measurement automation tools in part of the list of characteristics to collect is performed according to the measurement model.

At the "Load feeding" stage, the automation tools perform the load feeding procedures at the entry points specified for each type of load in the system model. The load is extracted from a pre-prepared database of test data and is automatically fed according to the schedule (the law of distribution over time) specified for each type of load in the load model.

At the "Data collection" stage, an automated collection of the values of the measured characteristics is performed. Measurement tools obtain the points of collection of these values from the system model, and the composition of the characteristics to measure it obtain from the measurement model.

At the "Interpretation and analysis of results" stage, all four models are used by automation tools. The requirements model is used to compare experimental results with system requirements. The load model and system model provide a report on the conditions of the experiment. Based on the measurement model, the comparison of the experimental results with the evaluation criteria is automatically performed.

According to the described technique, about three hundred full-scale load experiments were carried out, which made it possible to detect several dozen system errors. Based on the results of load testing, several dozen improvements to the systems were initiated aimed at eliminating signs of its degradation (decreased performance,

reactivity, violation of operating regulations). In a number of organization-owners of the AS, load testing is included in the release preparing schedule as an obligatory stage. The duration of the preparation of the load experiment decreased from 1-2 months to 1-2 weeks.

General characteristics of the automated load testing of the mission-critical banking system are given in table 2.

Table 2. Characteristics of composition, efficiency and intensity of use of the load tests

Tested system	**How many years operation is carried**	**Number of LOC (10^6)**	**Number of Functional Requirements (at high level)**	**Codes Developed since Maintenance (per year, %)**	**The volume of test data (the number of messages/documents) in running**	**The average number of running per year**	**Average quantity of found degradation signs/facts per year**
System 1	7	12	~ 600	~ 15	~ 4 000 000	**76**	5
System 2	7	5.5	~ 300	~ 15	~ 400 000	9	4
System 3	3	6	~ 400	~ 15	~ 8 500 000	**27**	9

7. Load testing infrastructure

Load testing is very costly, primarily because it requires a very expensive infrastructure, practically repeating the infrastructure of a system in operation. The creation of such an infrastructure is far from being possible for all system owners; these are rather exceptional cases than usual practice (Gotsutsov, 2017). A much more frequent situation is the use of the resources of the reserve capacities of a system in operation for carrying out load testing. Architecturally, reserve capacities are designed for hot reservation of system resources, and accordingly, load testing will consume resources intended to increase system availability. Therefore, it is very important to ensure a significant reduction in the transition time from functioning in the reserve mode to the load testing mode (occupation of resources) and the duration of the return to the initial state. In essence, we are talking about the automation of these processes. According to experience, only a list of operations for transition from one state to another takes 3-4 pages of printed text. Manual execution of these operations by the team preparing the

load experiment takes 1-2 hours and is associated with personnel errors. To overcome this drawback, it is necessary to develop programs - installers and uninstallers, which reduce the downtime of resources by several times, depending on the complexity of the infrastructure involved. Such programs can get rid of the technical errors of the staff.

8. Conclusion

The methods and tools described in this article have allowed over the course of 12 years to gradually increase the volume of regression tests by 3-4 times, while the number of testing specialists has not increased. Accordingly, the degree of coverage of the release of the AS with tests has also increased.

The number of residual errors in the release of the AS, that is, the number of critical errors that occurred in the first two weeks of operation of the release, does not exceed an average of three.

Load testing, as can be seen from table 2, allows to prevent the facts of degradation of the system, that is, it practically eliminates the risks of degradation of the AS during operation and large financial and reputation losses of the organization-owner of the AS.

The systematic application of the described methods of automated testing can significantly reduce the laboriousness and, in many cases, eliminate manual testing, as well as reduce the financial costs of testing and the timing of its implementation, while at the same time significantly, at times, increase the coverage of software by complex tests. As a result, it is possible to practically eliminate the risks of under-testing of software and guarantee the integrity of the release.

Automated load testing is a way to reduce costs when assessing the feasibility of fulfilling production regulations and a way of assessing the achievable performance of the system in case of load changes or system scaling.

References

Blokdyk, G. (2018). Software performance testing A Complete Guide, 5starcooks.

Dürr, G. (2010). Testing the Performance of Complex System Simulations, VDM Verlag.

Glavich, P., Farrell, C., Massey, C. (Ed.) (2010). .NET Performance Testing and Optimization - the Complete Guide, Red Gate Books.

Gotsutsov, S. (2017) Testing environment - technological platform for AIS PFR-2 life cycle management (in Russian), Actual Problems of System and Software Engineering 2017, CEUR-WS, vol.1989, 2017, pp.209-214

ISO/IEC 12207:2008 "System and software engineering - Software life cycle processes"

ISO/IEC 14764:2006 “Software Engineering - Software Life Cycle Processes – Maintenance”

ISO/IEC 15026-2:2011 Systems and software engineering - Systems and software assurance - Part 2: Assurance case

ISO/IEC/IEEE 29119-4:2015 “Software and systems engineering - Software testing - Part 4: Test techniques”

Klemm, B. (2013). Application Performance Testing: A Universal Performance Testing Methodology, pp.113, Kindle Series.

Lysunets, A.S., Pozin, B.A. (2013). Integrity Control Planning of Complex Automated Banking System by Methods of Functional and Performance Testing (in Russian), Programmnaya Ingeneria, 2013, no. 3, pp. 8—14.

Molyneaux, I. (2014). The Art of Application Performance Testing, 2nd ed., O′Reilly.

Pozin, B.A., Galakhov, I.V. (2011). Models in Performance Testing// Programming and Computer Software, 2011, Vol. 37, No. 1, pp. 15–25.

Pozin, B., Galakhov, I., Korotkov, A. (2017). The tool for XML-messages and messages packs generation for automated functional testing, Actual Problems of System and Software Engineering 2017. Vol. 1989. Aachen : CEUR Workshop Proceedings, 2017. p. 230-237.

Subraya, B.M. (Ed.) (2006). Integrated Approach to Web Performance Testing: A Practitioner's Guide, 1st ed, IRM Press Publishing. (pp. 1-368). IGI Global.

Zhen, M.J., Ahmed, E.H. (2015). A Survey on Load Testing of Large-Scale Software Systems // IEEE Transactions on Software Engineering (Volume: 41, Issue: 11, Nov. 1 2015), pp. 1091 – 1118.

Received November 6, 2018, revised April 18, 2020, accepted April 19, 2020