RESEARCH IN LOGISTICS & PRODUCTION

LOGISTICAL ASPECTS OF THE SOFTWARE TESTING PROCESS

Kazimierz Worwa*

* Faculty of Cybernetics, Military University of Technology, Warsaw, 00-908, Poland, Email: kazimierz.worwa@wat.edu.pl

Abstract The purpose of this article is characterization of the software testing process in terms of its logistical support. The software testing process is characterized as a complex, multi-stage project, with particular emphasis on its planning and organization. The role of logistical support and its importance for the efficiency of the testing process realization is highlighted. It was noted that proper software testing process planning and organization significantly affects the time and cost of that process.

Paper type: Research Paper

Published online: 30 April 2016
Vol. 6, No. 2, pp. 155-164
DOI: 10.21008/j.2083-4950.2016.6.2.5

ISSN 2083-4942 (Print)
ISSN 2083-4950 (Online)

Keywords: *software testing, software testing planning and organization*

1. INTRODUCTION

The main objective of the testing program is to detect and eliminate as many errors made in the course of the earlier stages of the program development process (IEEE Std 829-2008, 2008). The number of detected errors strongly depends on the scope, accuracy and organization of work related to the testing. The practice of software production shows that this work is very time-consuming and require considerable financial expenses. This ensures that the test phase is characterized by a very considerable share in the overall manufacturing cost of the program. In the case of large and complex software system the testing cost can constitute 40-70% of the total cost of its production (Pham, 2006). The realization of software testing phase, particularly with respect to complex software systems, is a multi-step and time consuming process. It is clear, therefore, that the successful implementation of the testing phase, particularly in relation to complex software systems, requires proper preparation and logistical support, as in the case of complex technical projects implementation. The purpose of this article is a characterization of the software testing process in terms of its logistical preparation.

2. ORGANIZATION OF THE SOFTWARE TESTING PROCESS

Software testing process includes: multiple program run with a prepared set of test cases, which is usually defined subset of all possible sets of input data; evaluating the results of the each run for specific test case by comparing the results with those expected; identifying causes of inconsistent with the expectations of the behaviour of the software and error location and debugging. The testing of a complex software system is generally a multistage process. Depending on the internal structure and intended use of the software under test, this process can include (Kit, 1995): low-level testing, including an unit testing and an integration testing and high-level testing, which may include: usability testing, function testing, system testing and acceptance testing.

2.1. Low-level testing

Low-level testing of the software system is executed with respect to its individual components or groups of components, created by their particular combination. In practice of software engineering these components are often modules, where module is a part of larger structure, eg. program, which may constitute a separate design and implementation task. Low-level testing requires a thorough knowledge of the internal structure of the tested software, including the so-called logic

of modules, and therefore is most often carried out by own design and implementation team. In practice, low-level may consist of unit testing and integration testing.

Unit testing is generally the first step of the testing process. It consists of an autonomous, i.e. implemented in isolation from the remaining components, testing each of the constituent component of the program. The main purpose of unit testing is to detect all possible errors that cause a variety of established, i.e. as defined in the requirements specification, the behaviour of the component under testing.

Integration testing is the process of planned merging separate components of the product and testing so formed groups of these components. The main purpose of integration testing is to detect and remove so called interfaces errors between components. In practice, depending on the type and structure of the software product under the integration testing this process can include several levels of integration. Testing integration performed for each level of integration may be a multistep process, wherein the number of steps is dependent on the number of components the component level testing and how they are combined in groups of components. Depending on the method of joining together (integrating) of the individual components the integration testing may take place as an incremental or nonincremental integration process.

Incremental integration testing relies on successive testing, in a certain way increased, groups of components to obtain the whole software product, where an expanding of the components group is to join to previously created and tested group a new program component. Depending on the order of test components, and the way of combining them into group integration testing can be realized as top-down testing or bottom-up testing.

Bottom-up testing involves combining and testing of components in the direction from bottom to top. The testing process begins by individual testing the terminal components, i.e. such components which do not activate other components. Then to the each terminal components are attached components which are directly activate by them, and so formed components are testing in the next stage. Joining the components from a higher levels of the component hierarchy to increase next component group, and then testing it ends after connecting and testing the latter component, ie. occupying the highest position in the component hierarchy. Bottom-up testing of each created in this process components group requires the use of special control modules, replacing missing, i.e. not included yet, the overarching components. The main task of the control module is to transmit to the component group under testing sets of input data. Of course, as the proceedings of the bottom-up testing process, artificial modules are replaced by the real components that take over their functions. The need to prepare and use these control modules may significantly affect the duration and cost of integration testing process, which is a disadvantage bottom-up testing.

Top-down testing involves connecting and testing the individual components in the direction from top to bottom. The procedure begins by testing the component occupying the highest position in the hierarchy of program components. After test-

ing the highest component there are attached the components activated by it directly (from lower hierarchy), and the process testing so formed components group is repeated. This process continues until join and test all the components, including terminal components. In practice, the incremental integration testing is usually carried out by methods that are specific combination of the bottom-up and top-down testing aimed at discounting the advantages of both methods and eliminating their disadvantages.

Nonincremental integration testing involves the simultaneous merging all program components and further testing of the whole software, wherein individual testing of each component is needed earlier. This method, compared with incremental integration testing methods, has a number of serious disadvantages. These disadvantage are mainly due to very limited opportunities to test the relationship between the individual components. However, this method is most commonly used in practice, which is mainly due to its simplicity.

2.2. High-level testing

The essence of high-level testing is to test the whole entire software product in order to assess the compatibility of its use conditions, the degree of required functions implementation and software behaviour to the requirements specification. To ensure full objectivity of the testing process, proper interpretation of results and formulated on their basis opinions high-level testing should be performed by a team that is fully independent from the team that produced the software under investigation. The practice of software engineering identifies four stages of advanced testing: usability testing, functionality testing, system testing and acceptance testing.

Compliance assessment of a software behaviour with requirements specification, including the requirements of the user, is the main goal of usability testing. The results of software usability testing should, among others, allow an assessment of: the availability of particular software functions, methods of entering input data and methods of sharing the results of the software work, easiness of use of the software by the user, easiness of learning the proper use of the software functions, completeness and accessibility of the help system.

The primary objective of functionality testing is to assess the compliance behaviour of the software in all its functions carried out, according to the requirements specification. The essence of a particular function testing is repeatedly run the software using the input data sets, resulting in activation of this function, i.e. activation of the software components realizing it. Effort and time-consuming testing process specific software function strongly depends on the type, extent of areas of its input and its complexity.

The purpose of system testing is to assess the compliance behaviour of a complete, fully integrated software, in conditions similar to those of its real operation, with the requirements and objectives set out in the specification requirements.

From the practice of software engineering it is known that system testing is both the most difficult step in the implementation of testing as well arousing much confusion as to its objectives and essence (Kit, 1995). System testing should be performed on a set of test cases created in a base of a deep understanding of the conditions, circumstances and method of using the software, ie. from the point of view of the user or customer. System testing is used to evaluate the behaviour of the whole software, including the degree of realization of its objectives as well as any required system documentation.

Acceptance testing is the last step in the process of software testing. Its aim is to assess the compatibility of the software system behaviour in real conditions of its usable life according to the requirements specification. Acceptance testing is performed by the user, usually in the form of a preliminary exploitation, lasting some period of time as determined in the requirements specification. Conclusions resulting from the use of a software system during its preliminary exploitation are the basis for the evaluation of the results of his work according to expectations, described in the requirements specification.

3. PLANNING THE SOFTWARE TESTING PROCESS

The implementation of each of the listed in the previous section stage of software testing process requires: determining the test plan; design and preparation of test data set; development of testing procedures, defining the list and how to implement actions required to carry out testing with separate test data sets; perform testing, i.e. run software product under testing (module, module group, program, program group, software system) for each test case from a set of test data; evaluating the results of testing by comparing the expected and the received results of the performance software product for each test case; locate and remove any detected errors.

Providing high efficiency of the software testing process requires a methodical, engineering approach to its implementation. This requires careful and realistic planning this process, the outcome of which is the plan of testing. According to (IEEE Std 829-2008, 2008) test plan is a document defining the subject, objectives and tasks of testing, the scope and method of testing, the means and the tools required to secure its implementation and the timetable for implementation of particular tasks comprising the software testing process. Because of the complex multi-step organization of the testing process, scheduling process is also generally consists of several stages. In practice, planning the implementation of the software testing process includes preparation (IEEE Std 1012-2004, 2005) the main test plan, setting out the aims and objectives of the whole process of software testing, identifying stages of the process and determining the scope and manner of their implementation, as well as drawing up a separate plan testing for each stage of the testing process, mentioned in the main plan. The main test plan, defining the whole

process of software testing, including the components of this process, should include (IEEE Std 829-2008, 2008): definition of tasks (including steps) and the objectives of the testing process; description of the methods and identification procedures for the implementation of each of particular task to achieve the objective of testing; the characteristics and conditions of the input data required for the proper implementation of each of the separate tasks of the testing process and output data resulting from the implementation of these tasks; timetable for the implementation of specific testing process stages, the deadlines for commencement and completion; identifying the needs for executive team, architectural hardware and software, special tools, etc., required for the proper implementation of specific testing; identification and description of possible difficulties and risks which may arise during the implementation of specific testing process stage, together with a description of proposed activities that will be required if these difficulties or threats are appeared; a description of the organizational aspects of the implementation of particular testing, tasks, including the identification of responsibility for the results of their implementation.

As mentioned previously, the scope and manner of implementation of each particular testing process stages determine the testing plans, prepared separately for each stage. Recommendations for layout and content of such a plan includes standard (IEEE Std 829-2008, 2008). The basic problem that must be solved at the stage of planning and preparation software testing process is to determine such a subset of the set of all possible input data sets that maximizes the probability of detecting all errors committed in the earlier stages of the software development process. This problem occurs due to the fact that, in general – because of the duration and cost of the testing process – it is impossible to test software based on the full set of all possible test cases. The problem of determining mentioned subset of the whole set of program input data is a problem of designing a set of test cases. Each test case is an acceptable combination of values that can accept input variables of tested software product required for his single run, wherein the input variables are those program variables whose values are determined directly on the basis of input data, eg. as a result of entering data to the program.

A method of determining a set of test cases on the basis of which software testing process is realized depends on the assumed test methods, each of which is based on a specific selection criterion of the test cases set. In addition to the method of forming the test cases set significant role for the results of software testing process and the amount of incur the time and financial expenditures, very important is the cardinality of test cases sets used in the different stages of the testing process. There is possible to determine an optimal cardinality of test cases set as a base for program testing process realization by solving a formal optimization problem, eg. with the objectives of maximizing program reliability and minimizing testing cost. Determining a set of test data used in the process of testing complex software systems is a very difficult and time-consuming activity that requires a very high regularity at work. In most cases of testing of complex software systems satisfactory results can only be achieved by using computer-aided

tools in test data generation process. The cardinality of the test case set is closely related with the assumed criteria for the completion of the testing process.

In the practice, depending on the approach to the problem of determining a set of test cases testing methods can be divided into two following classes: deterministic methods and random methods.

3.1. Deterministic methods

Due to the practical infeasibility of testing complex software system for the whole set of all possible input data, the process of checking and evaluating the behaviour of the tested software is substantially always deficient and incomplete. The only practical method of counteracting the negative effects resulting from the incompleteness of the test cases set used in the testing process is a such way of tests selection that guarantees a maximal probability of detect all errors (Pham, 2006).

Deterministic methods of designing a test cases set are based on an analysis of the requirements specification or on an analysis of the source code of tested program. In practice, depending on the subject of this analysis, there are two basic philosophies of test cases set design (Kit, 1995): functional design (called the black box method) and structural design (called the white-box method).

In the practice of software engineering the designing test cases set based on software requirements specification is realized with using some, in a certain way different methods, among which the most commonly used are: the method of equivalence partitioning and the method of cause-effect graphing.

Mentioned inability to testing the program on the whole set of its all possible test cases entails the necessity of determining that its subset, which would include representatives of certain classes of tests. This idea is the basis of the method of equivalence partitioning. Determining a test cases set based on this method is to split the set of all possible test data for a number of equivalence classes, wherein each class is created in such a way that all the test cases belong to the same class are each equivalent in capacity to detect an error in the software being tested. This means that if a certain test case from some equivalence class detects an error, it should be expected that any other test from that class also detect this error. Similarly, if a particular test case does not detect any error, the other tests from the same class also should not detect any error in the software being tested.

Designing a test cases set based on the method of cause-effect graphing consists in transforming the requirements specification for a set of test data. A cause-effect graph is a means of expressing formal program requirements, described mostly in informal language. An additional benefit that comes from using the cause-effect graphing method – beside of principal objective, which is to define a test cases set – is the possibility of additional validation of requirements specification in terms of their completeness and cross-compliance. The cause-effect graphing method is based on a detailed analysis of software requirements specification, to determine

the relationship between the program input data (causes) and program activities and results (effects). Particular relationships are represented by the graph, which of these reasons is called cause-effect graph. Due to the fact that the analyzed program requirements specifications can be a large text document, it is convenient (eg. to reduce the complexity of the constructed graph) to divide them into fragments, eg. that describe individual functions.

Determining a test cases set based on the source code of the program under testing, depending on the assumed degree of "cover" the logical structure of the program by created test cases set, may be implemented in practice on the basis of: statement coverage method, decision coverage method, logic condition coverage method, decision/condition coverage method and a multiple condition coverage method.

The statement coverage method consists in accepting such a test cases set which guarantees at least once activation of an each instruction in source code of the tested program.

In the decision coverage method the set of test cases is determined in such a way that the execution of all test cases belonging to that set guarantees at least once activation of each decision for each its logic value, that is to TRUE and FALSE values. The term "decision" means any conditional statement in program source code, whose execution causes a certain change of control in program execution. In general, each conditional instruction in program source code (except for unconditional jump instructions) contains a logical condition (predicate), whose value defines the mode of action of this instruction.

The decision coverage method the test cases set is requested in addition to guarantee both coverage of all conditional instructions (for each predicate values defining them) and coverage all other instructions.

In the logic condition coverage method test cases set is constructed in such a way as to ensure at least once activation of every logic condition of all conditional instructions in program source code, for each of the possible its logical values, that is to TRUE and FALSE values, while execution of each instruction in the test program. Constructed in this way test cases set must also guarantee at least once activates each of the other program instructions. It should be noted that the requirement concerning the need to cover all instruction was included for the same reasons as discussed earlier in the decision coverage method.

The decision/condition coverage method is a natural extension of the previously discussed the logic condition coverage method. Test cases set constructed in accordance with this method should ensure at least once activation of every decision and every logical conditions which are a part of those decisions, for TRUE and FALSE values respectively and at least once activation of each of the other program instructions. Disadvantage of the criterion underlying this method is an inability to ensure the real activation of particular logic conditions for each of their possible logic values and a lack of certainty as to the possibility of occurring all possible combinations of logic values of these conditions within

each predicate. The cause of first of these shortcomings method is objective, because it stems from the way in which compilers translate the logical expression of high-level language into the resulting (machine) language. Previously mentioned shortcomings of the decision/condition coverage method result that both it and the previously described methods do not allow for a good testing logic expressions in the various conditional instructions of the program. For precise testing of these terms the multiple condition coverage method should be used, in which the test cases set is created in such a way as to ensure the occurrence of all possible combinations of logic values for each predicate conditions, and at least once activation of each program instruction.

3.2. Random methods

Method of generating a case test set on the basis of the draw is very commonly used in the practice of testing. This is mainly due to its simplicity (no need for eg. A very labor-intensive analysis of the logical structure of the tested program), a direct consequence of which is the low cost of the testing process and its high susceptibility to automation. Another feature of a random method that determines its wide practical use is its high efficiency, measured by the number of errors detected in the testing process. Tests carried out to compare the effectiveness of commonly used in practice methods of testing have shown that a wide class of software testing based on random generation of test case is more effective than other methods, wherein particularly good results are obtained by using random generation of test data based on a probability distribution described by the operational profile of the tested program for determining the likelihood of each test case in in terms of its actual operational use. On the basis of literature random testing analysis it can be done following classification random testing methods: random testing and partition testing.

The idea of random testing consists in using a set of test cases set, created by drawing (usually without replacement) consecutive its components from the whole test cases set of the tested program. The fundamental problem of that method is to determine the probability distribution, according to which according to which particular test cases are drawn. In the absence of appropriate conditions relating to the nature of this distribution a uniform distribution over the set of all program test cases is often assumed. Very often used in practice a variation of a random testing method is method which uses a test cases set also created on the basis of all input data of the tested program, but the drawing a successive program test case is based on a probability distribution defined by a program operating profile.

The other of the aforementioned class of random testing methods are methods in which a successive test cases are drawn from some subsets of the entire set of program input data. Described approach assumes the division of the set of all

possible test case into so-called program partitions before making the actual process of drawing individual test case. This method of generating a test case set for a testing process is used most often in the context of the methods of structural testing. Partitions, to which set of all possible test cases of the tested program is divided, are generally extracted in such a way that the its constituent activate some of defined logical path. As a consequence, the program testing process is performed based on a number sets of test cases (corresponding to the number of separate partitions), wherein the drawing particular test case from separate partitions usually takes place on the basis of a uniform distribution.

3. CONCLUSION

The article presents the characteristics of the software testing process in terms of its logistical preparation, ie. the testing planning and organization. The characteristics of software testing process as a complex, multistage activity was presented. The successful realization of software testing process, particularly in relation to complex software systems, is conditioned by its proper planning and preparation. Crucial stages of the testing process with particular emphasis on methods of designing a test cases test were characterized. In the presented characteristics of the various steps of the testing process the role of logistical support and its importance for the efficiency of the process, including the time and cost of this process were highlighted.

REFERENCES

IEEE Std 829-2008, (2008) IEEE Standard for Software and System Test Documentation.
IEEE Std 1012-2004, (2005), IEEE Standard for Software Verification and Validation Plans.
Kit E., (1995), Software Testing in Real Word. Improving the Process. ACM Press Books, London.
Pham H., (2006), System software reliability. Springer, Berlin.

BIOGRAPHICAL NOTES

Kazimierz Worwa is an Associate Professor at the Cybernetics Faculty of Military University of Technology in Warsaw. He is deputy dean for education. His research interests include software reliability modelling, software testing methods and software efficiency. He is the author and co-author of many scientific publications. His paper appear in numerous journals, e.g. *Control and Cybernetics*, *Polish Journal of Environmental Studies, Research in Logistics & Production* and proceedings of international conferences, e.g. *Depcos-Relcomex, Information Management* or *System, Modelling and Control*.