GIS Odyssey Journal

2025 • Volume 5 • Number 2

https://doi.org/10.57599/gisoj.2025.5.2.5

Kazimierz Worwa[1]

ORGANIZATIONAL AND TECHNOLOGICAL PROBLEMS OF THE GIS SOFTWARE TESTING

Abstract: The testing stage, creating great opportunities to verify and shape software reliability, significantly increases the cost of its production. The effectiveness of the work related to testing, expressed by the interdependence of the level of program product reliability and the cost of testing it, strongly depends on the adopted testing strategy, specifying the organization and scope of the work performed.
The paper attempts to present a general description of the software testing process complex software systems such as GIS systems, with particular emphasis on the organizational and technological problems associated with the implementation of this process. The paper contains a description of the organization of the GIS software testing process and a description of the technological aspects of its implementation.

Keywords: software testing, software quality, software engineering, initial testing, advanced testing

Received: 19 May 2025; accepted: 21 June 2025

[1] Military University of Technology, Faculty of Cybernetics, Warsaw, Poland, ORCID ID: https://orcid.org/0000-0002-8153-958X, email: kazimierz.worwa@wat.edu.pl

Introduction

Despite the constant development and improvement of design and implementation methods used in the practice of GIS system software production, their current level still does not provide a full guarantee of creating a complex software product completely free of errors. These errors, detected after a shorter or longer period of software use, expose the user to specific losses, depending on the nature and purpose of this software. The need to detect errors made during the implementation of individual stages of the software production process as early as possible and to prevent the migration of errors to subsequent stages of this process, forces the organization of periodic assessments of the correctness of the obtained results. In software engineering practice, such assessments are performed as part of verification and validation. The essence of verification is the analysis and assessment of the results of the implementation of individual design and implementation projects, which make up the software development cycle, performed on the basis of documentation of the work carried out. Software engineering practice distinguishes two forms of verification: review and inspection. A common feature of these forms of verification is the performance of group analysis and assessment. They differ in the purposes of the assessments and the degree of formalization of the organizational and implementation procedures used. Validation is the process of assessing a system or its component, which is in the final phase of its development cycle, in order to determine whether the requirements specified in the requirements specification have been met. Considering the current state of development of software engineering methods, such an assessment is most often made based on the results of running the assessed software for a specific set of input data sets and comparing them with the expected results. Such validation of a software product is therefore possible only when it reaches the stage of computer executability. Software validation is carried out in practice through its testing. The verification and validation processes are complementary. Although they are carried out using different methods and in different places of the software development process, they have the same main goal: to detect errors made in earlier stages and phases of this process. Verification enables the detection of errors in the early stages of the software production cycle. Properly organized and carried out, it can be an effective means of preventing the migration of errors to subsequent phases and stages of this cycle. Validation enables the detection of errors in software that is usually at an advanced stage of its production process. Given that most errors are made in the initial stages of the software production cycle (Kit, 1995; Roman, 2015), removing errors detected at this stage often requires changes to the software design or even the requirements specification. In this situation, it is obvious that the cost of removing errors detected as a result of validation procedures, i.e. testing, is usually much higher than the cost of removing errors detected earlier, e.g. as part of project verification. However, it should be clearly emphasized that the aforementioned high cost of removing errors detected in the final phase of the software production process is usually a much lesser evil (also in terms of cost) than their disclosure at the stage of software use. The results of the study of the effectiveness of detecting errors made at various stages of the software development

cycle, confirmed by practice, show that only about 20% of all errors detected in the entire cycle are revealed by verification methods, while about 80% of these errors are detected as a result of the testing stage (Kit, 1995; Roman, 2015). The given proportions clearly illustrate the importance of testing in the process of ensuring the required level of reliability of the produced software. The basic goal of program testing is to detect and remove as many errors made during the implementation of earlier stages of the production process as possible (IEEE Std 829-2008, 2008; ISO/IEC/IEEE 29119-3:2021, 2021). Detecting and removing a certain number of errors contributes to increasing the program's reliability. The number of detected errors strongly depends on the scope, accuracy and organization of the testing work. The practice of software production shows that this work is very time-consuming and requires high financial outlays. This results, among other things, in the testing stage being characterized by a very significant share in the overall cost of program production. In the case of a large and complex software system, the cost of testing can constitute 40-70% of the total cost of its production (Pham, 2006). Software engineering practice clearly indicates that the testing stage plays a very significant role in shaping the level of software quality of IT systems. Due to the fact that the basic form of verification of the achievement of the expected level of quality by the software being produced is the process of its testing, the issue of testing has a relatively rich literature, both domestic and international, within which selected problems are taken up regarding the organization and technology of implementing individual stages of this process. Representative examples of publications in this area can be works (Craig, 2002; Myers et al., 2004; Ammann & Offutt, 2017; Kit, 1995; Pham, 2006; Wiszniewski & Bereza-Jarociński, 2009; Roman, 2015; Pressman 2018). The implementation of the software testing stage, including GIS system software, is a multi-stage and labor-intensive undertaking. It is therefore obvious that the successful implementation of the testing stage, especially in relation to complex software systems, requires proper preparation and logistical support, similarly to the implementation of complex technical undertakings. Planning the testing process requires, in particular, determining the organization of work that makes up the testing stage and the method of generating the used set of test data sets (tests) and its size. Decisions made during testing process planning have a fundamental impact on the time and cost of this process.

The aim of this article is to present a general description of the GIS software testing process, with particular emphasis on the organizational and technological problems associated with the implementation of this process. The remaining part of the work contains a description of the organization of the GIS software testing process and a description of the technological aspects of its implementation.

Material and methods

Organization of the GIS system software testing process. The basic goal of software testing is to detect errors made in earlier stages of its development cycle, causing incorrect behavior of the software, i.e. inconsistent with the requirements specification. The essence of software testing is:

- multiple execution of the software for a prepared set of test data, which is usually a specific subset of all possible sets of input data,
- evaluation of the results of execution of the tested software for individual sets of data, by comparing the obtained results with the expected ones,
- identification of the causes of the software behavior inconsistent with expectations and localization and removal of errors.

The process of testing a complex software system is usually a multi-stage undertaking. Depending on the internal structure and purpose of the software being tested, this process may include (Kit, 1995; Craig, 2002; Myers et al., 2004; Pressman, 2018):

1) initial testing:
   - individual testing,
   - integration testing;
2) advanced testing:
   - usability testing,
   - function testing,
   - system testing,
   - acceptance testing.

Initial testing. Initial testing of a software system is performed in relation to its individual components or groups of components, created by their specific combination. In software engineering practice, these components are most often modules, which are fragments of larger structures, e.g. programs, which can constitute independent design and implementation tasks. Initial testing requires a thorough knowledge of the internal structure of the tested software, including the so-called module logic, and is therefore most often performed by the design and implementation team itself. The principle that a programmer should not test his own module or program should be observed. The essence of testing is to find errors in the software, while psychological conditions make it very difficult for most programmers to adopt a destructive attitude towards the results of their own, often several months, work. Good results, on the other hand, are achieved by entrusting the testing of a specific module to the author of the module cooperating with the tested module, e.g. the author of the module invoking the tested module. Individual testing is usually the first stage of the testing process. It consists of autonomous testing of individual program components, i.e. performed in isolation from other components. The main goal of individual testing is to detect all possible errors that cause the tested component to behave differently than the one assumed, i.e. specified in the requirements specification. Testing a specific component in isolation from the rest of the program, including without other components cooperating with it, may require the use of:

- a control module, the main task of which is to call, i.e. initiate execution, with a specific set of input data and collect (print, record) the results of the tested component's operation;
- mock-ups of modules that simulate the operation (functions) of components that the tested component calls during its operation; in the case where the tested component is a so-called terminal component, i.e. does not call other components, mock-ups of other components are not needed for its testing.

Integration testing. Integration testing is a process of planned connection (merging) of isolated components of the software product under consideration and testing of such created groups of these components. The main goal of integration testing is to detect and remove errors in interfaces (couplings) between components. In practice, depending on the type and structure of the software product being produced, the integration testing process may include several integration levels.

Integration testing, carried out within each integration level, may be a multi-stage process, with the number of these stages depending on the number of components integrated at a given level and the method of combining them into component groups subject to testing. Depending on the method of combining (integrating) individual components, integration testing may proceed in an incremental or non-incremental manner.

Incremental integration testing consists in successive testing of a group of components, expanded in a specified manner, until the entire tested software product is obtained, whereby the expansion of the component group consists in adding a new component to the previously created and tested group. Depending on the order of testing components and the way they are combined into groups, in practice, the following are distinguished:

- bottom-up testing,
- top-down testing.

Bottom-up testing involves combining and testing components from the bottom up. The testing process begins with individual testing of terminal components, i.e. those that do not call other components. Then, components that directly call them are added to individual terminal components, after which the groups of components created in this way are tested. Adding components from the next, higher levels of their hierarchy and testing the group of components expanded in this way ends after adding and testing the last, i.e. the highest-positioned, component. Bottom-up testing of each of the component groups created in this process requires the use of special control modules, replacing the missing, i.e. not yet added, parent components in the tested group. The main task of the control module is to pass subsequent sets of input data to the tested group of components. Of course, as the bottom-up testing process progresses, the artificial control modules are replaced by real components that take over their functions. The need to prepare and use these control modules can significantly impact the duration and cost of the integration testing process, which is a disadvantage of bottom-up testing.

Top-down testing involves combining and testing individual components in a top-down direction. The procedure begins with testing the component occupying the highest (peak) position in the hierarchy of isolated components. After testing the top component, the components directly called by it (from lower levels of the hierarchy) are added to it, and then the testing process of the group created in this way is repeated. This process continues until all components, including terminal components, are added and tested. A comparison of both presented methods of incremental integration testing allows us to see their mutual advantages and disadvantages. The main advantage of top-down testing is the possibility of obtaining an early, albeit skeleton, but working version of the tested

software product. This allows for appropriately early implementation of subsequent testing stages, e.g. testing usability or software functions. One of the disadvantages of top-down testing results from the need to use, often expensive, mockups of missing components. Another disadvantage is the inability to test the added components sufficiently thoroughly. This is because this testing is carried out through a layer of components added earlier. In practice, such a situation may mean the need for prior, individual testing of all or some components. Bottom-up testing usually does not provide the possibility of obtaining a working skeleton of the entire software product early on, which is its main disadvantage. Another disadvantage of this method is the need to use control modules that enable testing. An undoubted advantage of bottom-up testing is the possibility of thoroughly testing the internal logic of the added components, which often allows for their prior individual testing to be omitted. In practice, incremental integration testing is carried out using methods that are a specific combination of top-down and bottom-up testing, aimed at discounting the advantages of both methods and eliminating their disadvantages. The simultaneous use of top-down and bottom-up testing leads to mixed testing and consists in combining the integration processes from the top and bottom, until they "meet" in the middle of the structure created by the separated components. The meeting point, depending on the structure of the tested software, should be determined before starting integration testing. Mixed testing is used in large and complex software systems. Non-incremental integration testing involves the simultaneous integration of all isolated components and further testing of the entire software. This requires prior, individual testing of each component. This method, compared to incremental integration testing methods, has a number of serious disadvantages. These disadvantages result primarily from very limited possibilities of testing the connections between individual components. However, it is most often used in practice, which is mainly due to its simplicity. This method can be useful in the case of testing small programs with a low-complexity modular structure.

Advanced testing. Advanced testing involves testing the entire, complete software product in order to assess the compliance of its use conditions, the degree of implementation of the required functions and its behavior with the requirements specified in the requirements specification. In order to ensure full objectivity of the testing process, interpretation of the obtained results and assessments formulated on their basis, advanced testing should be carried out by a team independent of the team that created the software in question. Software engineering practice distinguishes four stages of advanced testing:

- software usability testing,
- software function testing,
- system testing,
- acceptance testing.

Assessing the compliance of the software's behavior with the requirements specified in the requirements specification, including user requirements, is the main goal of usability testing. The results of software usability testing should, among other things, enable the assessment of:

- the availability of individual software functions,
- the method of entering input data and sharing the results of software operation,
- the ease of using the software by the user,
- the ease of learning how to properly use the software by the user,
- the completeness and availability of the help system.

The basic goal of software function testing is to assess the compliance of the software's behavior, within the scope of all the functions it performs, with the requirements specification. Software function testing therefore focuses on assessing the correctness and accuracy of the results of the implementation of individual functions performed by the tested software. The essence of testing a specific software function is to repeatedly run the software using sets of input data, which cause the activation of this function, i.e. activation of the software components that perform it. The laboriousness and time-consuming nature of the process of testing a specific software function strongly depends on its type, the scope of its input data domain and its complexity. Achieving the goals set for software function testing depends to a decisive degree on the proper design and preparation of the test data set, based on which the testing is performed.

The goal of system testing is to assess the compliance of the behavior of complete, fully integrated software, in operating conditions similar to the conditions of its actual use, with the requirements and goals specified in the requirements specification. Software engineering practice shows that system testing is both the most difficult stage of testing to implement and the one that raises the most misunderstandings about its goals and essence (Kit, 1995). System testing should be carried out based on a set of test data created based on a deep understanding of the conditions, circumstances and ways of using the software, i.e. from the point of view of its user or client. System testing is used to assess the behavior of the entire software system, including the degree of implementation of the assumed goals, as well as any required user documentation. Unlike the earlier stages of testing, the test data sets used in the system testing process rarely concern the invocation of a single software function. This data usually takes the form of scenarios describing specific sequences of operations performed by the software system being tested.

In the process of assessing the compliance of the actual and assumed behavior of the software system, which is the essence of system testing, the following aspects of this behavior should be taken into account:

- responding to so-called overloads, resulting from short-term, high load of the tested software system, resulting from, for example, a large number of transactions reported for service in a short period of time in the banking system, from a large number of reports appearing simultaneously in the seat reservation system, from the appearance of a large number of signals requiring immediate response in the system controlling the technological process, etc.;
- responding to large amounts of input data, constituting a long-term load on the tested system;
- the level of data protection and security, including in particular the effectiveness of software data protection mechanisms, preventing, for example, access to specific data and programs by unauthorized persons;

- system efficiency, defining the results of the system's work in categories such as throughput, efficiency, response time, time of handling a single report, etc.;
- the level of resource consumption required to ensure proper software operation, including in particular the occupancy of RAM and external memories, the occupancy of external devices used, etc.;
- hardware and software compatibility with the immediate environment;
- response to failures or unexpected behavior of elements of the hardware used, cooperating software systems or specific components of the software being tested;
- correctness of the required diagnostic software, including memory content analysis programs and tracking programs, which are a tool supporting the software maintenance process, e.g. facilitating the location and removal of errors that may occur during the operational use of the software;
- correctness and ease of implementation of software installation procedures in the actual work environment;
- completeness and substantive level of the required operational and maintenance documentation;
- compliance of the reliability indicator values specified in the requirements specification with the actual values of these indicators; although the assessment of this aspect of the software system operation in system testing conditions may be difficult (e.g. how to check whether the mean time between failures is 2000 hours?), it cannot be omitted.

Acceptance testing is the last stage of the software testing process. Its purpose is to assess the compliance of the software system's behavior in the actual conditions of its operational use with the requirements specified in the requirements specification. Acceptance testing is performed by the user, most often in the form of trial operation, lasting a fixed period of time specified in the requirements specification. Conclusions resulting from the use of the software system during its trial operation are the basis for assessing whether the results of its work are consistent with the expectations expressed in the requirements specification.

Software testing process technology. The implementation of each of the software testing stages listed in the previous section requires:

- defining a testing plan,
- designing and preparing a test data set,
- developing testing procedures that specify the list and method of performing the activities required to perform testing using the separated test data sets,
- performing testing, i.e. launching the tested software product (module, group of modules, program, group of programs, software system) for each data set from the test data set,
- evaluating the testing results by comparing the expected and actual results of executing the tested software product for each test data set,
- locating and removing any detected errors.

Planning the testing process. Ensuring high efficiency of the software testing process requires a methodical, engineering approach to its implementation. This requires

accurate and realistic planning of the implementation of this process, the result of which is a test plan.

According to IEEE Std 829-2008 (2008) or ISO/IEC/IEEE 29119-3:2021 (2021), a test plan is a document defining the subject, goals and tasks of testing, the scope and method of testing, the means and tools required to secure its implementation and the schedule for the implementation of individual projects that make up the testing process. Due to the complex, multi-stage organization of the testing process, planning this process usually also consists of several stages. In practice, planning the implementation of the software testing process includes the preparation of (IEEE Std 1012-2004, 2005):

- a master test plan, defining the goals and tasks of the entire testing process of the software in question, identifying the stages of this process and defining the scope and method of their implementation;
- a separate test plan for each stage of the testing process, separated in the master plan.

The master test plan, defining the organization of the entire software testing process, taking into account the component stages of this process, should include (IEEE Std 829-2008, 2008; ISO/IEC/IEEE 29119-3:2021, 2021):

- definition of tasks (including stages) and objectives of the testing process;
- description of methods and identification of procedures for the implementation of each of the separate tasks, in order to achieve the assumed testing objective;
- characteristics of input data and conditions required for the correct implementation of each of the separate tasks of the testing process and output data resulting from the implementation of these tasks;
- schedule for the implementation of individual tasks of the testing process, specifying their start and end dates;
- definition of needs in terms of the executive team, hardware and software architecture, specialist tools, etc., required for the proper implementation of individual testing tasks;
- identification and description of possible difficulties and threats that may occur during the implementation of individual tasks of the testing process, together with a description of the proposed actions that will be required in the event of the occurrence of such difficulties or threats;
- description of the organizational aspects of the implementation of individual testing tasks, including the definition of responsibility for the results of their implementation.

In software engineering practice, the master test plan is usually part of the software verification and validation plan, the layout and content of which are specified by the standard (IEEE Std 1012-2004, 2005). The software verification and validation plan defines all activities related to the verification of the correctness of the implementation of individual stages of the software development cycle and the phases that make them up, including the implementation of the software testing process. The activities included in the software verification and validation plan are focused on:

- the earliest possible detection and removal of errors made in individual stages of the software development cycle;
- reducing the risk of exceeding the planned time and financial outlays;

- improving the level of software quality, including increasing its reliability;
- increasing the effectiveness of managing the implementation of the entire software development process.

The software verification and validation plan is one of the documents specified by the software quality assurance plan, the execution of which – in accordance with the standard (IEEE Std 1012-2004, 2005) – is mandatory in the production process of the so-called responsible software. The software quality assurance plan is the highest (in the hierarchy of planning documents related to the software production process) plan, containing direct references to the software testing process. It contains a description of all activities, used standards, measures and indicators, as well as tools and means for achieving the required level of quality by the produced software. The layout and content of the software quality assurance plan are specified by the standard (IEEE Std 1012-2004, 2005). In accordance with the requirements specified by this standard, the remaining mandatory documents are: software requirements specification, software project description, software verification and validation plan implementation report and user documentation.

The execution of a software quality assurance plan, in accordance with the requirements specified by the standard (IEEE Std 730.1-1995, 1995), is mandatory for responsible software and recommended for other software.

According to the previous comments, the scope and method of implementation of each of the separate stages of the testing process are determined by test plans, prepared separately for each stage. Recommendations regarding the layout and content of such a plan are included in the standard (IEEE Std 829-2008, 2008; ISO/IEC/IEEE 29119-3:2021, 2012). According to this standard, the test plan should specify:

- the subject of testing, being a specific subset of the components of the software being produced, with an indication of the required – as input data – documents, including requirements specifications, design specifications and user documentation (in the scope concerning the subject of testing);
- the purpose of testing, with an indication of the detailed characteristics and properties of the tested software, subject to verification and evaluation;
- the method of testing, with particular consideration of the activities, methods and tools required to verify and evaluate the indicated characteristics and properties of the tested software;
- criteria for completing testing, describing the conditions that must be met in order to consider testing as completed;
- criteria and method for assessing the results of running the tested software for individual test data sets;
- a list of detailed tasks, so-called testing tasks, the execution of which is required for the proper preparation and implementation of the testing process;
- a list of documents that should be the result of the preparation and implementation of the testing process;
- detailed needs in terms of computer hardware and software required to secure the implementation of the testing process and the conditions of its use, taking into account, among others, data protection conditions;

- organization and composition of the team performing the testing, taking into account the division of competences and responsibilities;
- a schedule for the implementation of the testing process, the construction of which must take into account the start and end dates of individual testing stages, included in the main testing plan;
- possible difficulties and obstacles that may occur during the implementation of individual tasks of the testing process, with a description of the proposed actions and remedies.

Results and discussion

Research results. In accordance with the previous remarks, the significance of the software testing stage in the process of ensuring a high level of its reliability is very high. This results, among others, from the fact confirmed by practice that as many as approx. 80% of all errors made in the entire software development cycle are detected as a result of the testing stage (Kit, 1995, Roman, 2015). The considerations presented clearly indicate that the implementation of the software testing process is a complex, multi-stage, time-consuming and labor-intensive undertaking. This problem is particularly important in relation to complex software systems, such as GIS system software. Successful implementation of the testing process of such software systems requires proper preparation and logistical security, with particular emphasis on proper planning of this process, including determining the organization of work that makes up the testing stage and the method of generating the used set of test data sets (tests) and determining its size. Decisions made during the planning of the testing process have a fundamental impact on the time and cost of this process.

Testing practice shows that testing efficiency can be increased by, among others:

- using appropriate tools and means to support the implementation of the software testing process,
- using standards,
- properly documenting the implementation of the work carried out.

Using appropriate tools to support the implementation of the software testing process makes this process easier and more effective. Using these tools is currently an important element of testing technology. The growing importance and popularization of the practice of using tools to support the implementation of the software testing process occurred as one of the direct consequences of the constantly expanding area of application of computer systems and the accompanying increase in quality and reliability requirements for their software. Computer-aided implementation of the software testing process, and especially automation of this process, enables, among others:

- increasing the number of detected errors, by ensuring greater systematicity in the implementation of the adopted testing plan,
- shortening the duration of the testing process and reducing its cost, by minimizing the number of test data sets used, while ensuring the implementation of the assumed testing goals.

In software engineering practice, many different computer-aided software testing (CAST) tools are used, which are used in different phases of this process. These tools support:

- planning the testing process,
- designing test data sets,
- executing the tested software for subsequent test data sets and evaluating the results of these executions.

The means and tools for computer-aided design of the test data set constitute the most promising group of tools for increasing the effectiveness of the testing process. Within this group of means and tools, we can distinguish:

- automatic test execution systems,
- test effectiveness analyzers,
- tracking programs,
- hardware and software simulators.

The basic problem that must be solved at the stage of planning and preparing the software testing process is to determine such a subset of all possible input data sets that would maximize the probability of detecting all errors made in the earlier stages of the software development process. This problem arises due to the fact that, in general – due to the duration and cost of the testing process – it is impossible to test software based on the entire set of all possible sets of input data. The problem of determining the aforementioned subset is the problem of designing a set of test data (tests). Each test is an acceptable combination of values that can be assumed by the input variables of the tested software product, required for its single launch, whereby input variables are variables whose values are determined directly on the basis of the input data, e.g. as a result of executing the data loading instruction. The method of creating a set of test data, based on which the software testing process will be carried out, depends on the adopted testing method, each of which is based on a specific criterion for selecting test data sets. In addition to the method of selecting individual test data sets, the numbers of test sets used in individual stages of the testing process are of decisive importance for the testing results and the amount of time and money incurred during it.

Discussion of the results. Among the implementation phases of each stage of the testing process, the most important – due to its impact on the course and results of the work carried out – is the test data set design phase. The test data set, which is the basis for the implementation of the testing process, determines the degree to which the assumed goals are achieved, the course of the work carried out, and the duration and cost of testing. Due to the existence of significant differences in defining the detailed testing goals for the individual stages of the testing process, the design of the test data set for the implementation of these stages can be carried out using methods based on both the functional design philosophy and the structural design philosophy. Based on practical experience, general suggestions can be made regarding the usefulness of functional and structural design for defining test data sets used in the individual stages of the software testing process. From these experiences, particularly good results are achieved by using:

- structural design methods to determine test data sets for individual testing,
- functional and structural design methods to determine test data sets for integration testing,
- functional design methods to determine test data sets for usability and software function testing, system testing and acceptance testing.

Proper determination of the test data set has a very significant impact on the effectiveness of each stage of the software testing process, with this effectiveness being determined by the degree of achievement of the assumed goals and the level of time and financial outlays incurred.

It is worth emphasizing that the basic goal of software testing, which is to detect as many errors as possible, made during the implementation of earlier stages of the software production cycle, is achieved in practice under conditions of limited time and financial outlays. In the process of planning and implementing the testing process, solutions must therefore be preferred that ensure the implementation of the intended goals with the least amount of time allocated for testing and at the lowest possible cost. This is achieved through consistent use of recognized standards, as well as the use of appropriate computer-aided tools for the implementation of the testing process. The greatest impact on the amount of time and financial outlays incurred in the process of implementing each testing stage, distinguished in the plan, is the size of the test data set used. It should be emphasized that the form and content of the test data set is in practice a certain compromise between the need to confirm that the tested software has achieved the required level of reliability (from this point of view, the number of tests should be as large as possible) and the need to minimize the amount of time and financial outlays incurred (from this point of view, the number of tests should be as small as possible).

Conclusions

The article presents the characteristics of the GIS system software testing process. The characteristics of the software testing process are presented as a complex, multi-stage undertaking, the successful implementation of which, in particular with regard to complex software systems like GIS systems, is conditioned by its proper planning and management. The basic stages of the testing process are characterized, with particular emphasis on the methods of designing a set of test cases. The presented characteristics of the testing process emphasize the role and importance of its proper logistical support, which allows for a significant increase in the effectiveness of the work carried out in this area, especially in terms of the duration and cost of its implementation.

It follows from the presented considerations that the software testing stage, closing its production cycle, creates the last opportunity to detect design and program errors that were made during the implementation of earlier stages of this cycle. The implementation of the software testing stage, in particular with regard to complex software systems such as GIS systems, is a multi-stage and labor-intensive undertaking. It was emphasized that the successful implementation of the testing stage, especially in relation to complex software systems, requires proper preparation, planning, monitoring of implementation

and ensuring appropriate logistical security. The proper course of the software testing process is conditioned by its proper management. Due to the fact that testing is one of the stages (along with defining requirements, designing and programming) of the software development process, the method of managing the implementation of the testing stage depends on the adopted method of managing the entire software development process, while the practice of modern software engineering shows that in relation to large projects, highly formalized management methodologies are most often used, including primarily the RUP methodology, while for medium and smaller projects, agile methodologies are increasingly used, e.g. the XP methodology. Management of the software development process, both in relation to highly formalized methodologies and agile methodologies, covers such areas as: human resources management, financial management, time management, knowledge management, relationship management, including in particular customer relationship management, incident management, quality management and risk management.

Acknowledgements

This work was financed/co-financed by Military University of Technology under research project UGB 531-000023-W500-22.

References

Ammann P., Offutt J. (2017). Introduction to software testing. Cambridge University Press.

Craig R.D. (2002). Systematic Software Testing. Artech House.

IEEE Std 829-2008 (2008). IEEE Standard for Software and System Test Documentation.

ISO/IEC/IEEE 29119-3:2021 (2021). Software and systems engineering – Software testing. Part 3: 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. Addison–Wesley.

Myers G.J., Sandler C., Bagdett T., Thomas T.M. (2004) The art of software testing. Wiley.

Pham H. (2006). System software reliability. Springer.

Pressman R.S. (2018). Software engineering: a practitioner's approach. McGraw–Hill, New York.

Roman A. (2015). Software testing and quality. Models, techniques, tools. PWN, Warsaw.

Wiszniewski B., Bereza-Jarociński B. (2009). Teoria i praktyka testowania programów (*Theory and practice of program testing*). PWN, Warsaw.