UDC 004.41
DOI: 10.20535/SRIT.2308-8893.2025.3.05

USE OF METHODS AND TOOLS FOR ENSURING SOFTWARE QUALITY

A.S. SHANTYR

Abstract. This paper proposes an examination of effective methods and tools for ensuring software quality. The scope of this topic includes current issues related to software quality assurance within the context of analyzing methods and tools used in practice to develop high-quality software. During the modeling process, a new comprehensive model for software quality assurance has been developed, combining modular testing, integration testing, and continuous integration methods. The advantage of this development is its enhanced adaptability to addressing key challenges in software quality assurance. Based on the developed model, strategies and approaches are proposed to improve configuration management processes and identify vulnerabilities in software systems.

Keywords: comprehensive model, quality standards, integration testing, unit testing, technological challenges, integration testing, continuous integration.

INTRODUCTION

In the modern digital world, often referred to as the era of software, the concept of software quality has become not only relevant but critically important. In all areas of activity, from business and science to everyday needs, software has become an integral component. However, the mere existence of a software product does not guarantee its success [1]. In this regard, it is quite appropriate to agree with the views of the authors of the work [2], which conclude that quality becomes the decisive factor influencing decision-making and meeting user needs [2].

According to the authors of the study [3], achieving high-quality software requires not only innovative technologies but also effective methods and tools for ensuring quality. These methods and tools encompass a wide range of activities, from testing and code analysis to quality management and the implementation of best practices in the software development process. Scientific research in the field of software quality assurance has been conducted for several decades, and many aspects have already been addressed, including:

- test automation: numerous tools, frameworks, and Continuous Integration / Continuous Deployment practices have been developed for automatic code quality verification [1–3];
- quality metrics: a set of metrics has been defined and implemented, such as code coverage, system response time, the number of defects per unit of code, and others [4; 5];
- software security: immunity to many types of attacks has been developed, popular vulnerabilities such as SQL injection, cross-site scripting (XSS) and others have been identified and fixed;

- development methodologies: there are many recognized methodologies, such as Scrum, Kanban, DevOps, which regulate development and quality assurance processes.

However, there are also unsolved problems and directions for further research, namely:

- intelligent testing: unsolved questions regarding the evaluation of the development of machine learning methods for automated testing that are able to adapt to changes in the code and respond to new functions. The essence of the problem of intelligent testing within the framework of quality assessment of software systems boils down to the fact that testing requires an understanding of the functionality of the program and the possibilities of automation. Ensuring the adaptability of tests to changes in the code requires intelligent systems. The complexity of solving the above-mentioned problem lies in the fact that at the current level of technological development of qualitative evaluation of software systems, there are technological difficulties in the development of optimized machine learning algorithms that can adapt to changes in the software code and detect new functionalities automatically;
- security of microservice architecture: issues related to ensuring security and integrity in the context of the rapid development of microservice systems, given the large number of interacting components, have not been fully resolved. According to [5], the significant number of interacting microservices creates a vast attack surface, complicating the detection and resolution of potential vulnerabilities. As per [6], the challenge in addressing this problem at the methodological level lies in the incomplete implementation of mechanisms that address the development of systems for detecting and monitoring potential attacks, as well as the development of secure microservice development practices;
- automation of vulnerability detection: the issues surrounding the development of effective tools for automated detection of new vulnerabilities and their subsequent remediation have not been fully addressed. According to [7], the essence of this problem is that vulnerabilities evolve, and tools need to detect new security threats. The challenge here is the constant need to develop more effective, cutting-edge tools and technologies capable of identifying new and unknown vulnerabilities through contextual understanding and advanced detection methods;
- combination of functional and learning-based testing: approaches that combine functional testing with machine learning to more effectively detect defects have not been fully developed;
- security guarantees at large scales: the development of methods and architectures to ensure the security of software systems in large, distributed, and complex ecosystems has not been fully addressed. The core of the problem is that in large ecosystems, such as cloud services, ensuring security at all levels is challenging [8]. Additionally, there are difficulties in using distributed architecture, encryption, and security standards to protect vast amounts of data and systems;
- ethical aspects in testing: the development of standards and ethical norms for automated testing systems and their impact on people has not been fully addressed.

The core of the problem is the need to solve issues related to testing that may affect user privacy and security. The complexity of this issue lies in the definition of standards and the implementation of practices that ensure ethical standards in testing.

ANALYSIS OF RECENT PUBLICATIONS

Foidl H., Felderer M. [1] considered the principles of integration of software quality models based on risk-oriented testing. The authors noted above solve the problem of defining software by using quality models. Within the scope of the raised topic, the authors propose to integrate software quality models based on risk-oriented testing, according to their position, this approach will help significantly improve the effectiveness of testing, allowing to focus on areas with high risk for software.

The advantages of this approach are more efficient use of resources: focusing testing on high-risk areas allows for more efficient use of limited resources, directing them to the most critical aspects of the software. In addition, risk-based testing, based on quality models, allows you to better consider business needs and focus on aspects of the program that have the greatest impact on business processes. The authors also note that the integration of quality models helps to identify potential problems and risks in advance, which allows developers to pay more attention to their solution in the early stages of development. In addition, the authors proved that focusing on important areas of risk improves software quality by identifying and solving problems important to users. However, the integration of software quality models into risk-based testing proposed by Foidl and Felderer may have certain drawbacks, namely:

- complexity of integration: transferring quality models into the context of risk-based testing can be challenging, especially if there are different methods and approaches to defining quality and risks;
- data heterogeneity: quality models may be based on different data sets, and risks may arise from various sources. Integrating this data can be problematic due to its heterogeneity and incompatibility;
- ambiguity in defining risks and quality: the concepts of risk and quality may have different interpretations for different stakeholders, making the integration of models difficult due to this ambiguity;
- need for large amounts of data: effective operation of integrated models requires a significant amount of data, both on software quality and risks. This can be a problem where access to such data is limited;
- system complexity: integrating models may require significant effort and resources to develop, implement, and maintain complex systems.

Thus, while integrating software quality models into risk-based testing can bring substantial benefits in identifying high-risk areas for software, it is essential to carefully study and consider the aforementioned challenges for successful application of this approach. K. Sahu and R. K. Srivastava [2] examined the foundations of software error prediction using neuro-fuzzy logic methods. Their research emphasizes software reliability and the need to find effective error prediction methods. However, the researchers did not explore the possibility of integrating error prediction methods with other quality assurance methods, such as statistical models, testing, or defect analysis. In general, recent research in the field of effective methods and tools for software quality assurance significantly improves the efficiency of quality assurance processes. According to [9], at the technological-methodological level, the issues related to the aforementioned analysis require a deep understanding of the functional and technical aspects of software systems, as well as innovative approaches to solving them. The complexity lies in combining technical expertise with creativity and flexibility in addressing diverse problems. According to [10], research on software quality assessment methods is ongoing to identify effective tools and techniques. However, according to [10], research in the field of software quality assessment requires continuous improvement of methods and tools, as well as consideration of changes in the technological landscape and user requirements. According to [11], research on effective methods and tools for software quality assurance is a crucial field in information technology. According to [12], issues related to software quality assurance can arise for various reasons, and solving them requires a systematic approach. In the framework of research on quality assurance methods and tools, the researchers from works [1–15] encountered numerous issues requiring additional comprehensive analysis. Specifically, in the area of automated testing, it is advisable to conduct research to develop the most appropriate automated tests covering various aspects of the software and its functionality to prevent malfunction in real-world conditions [1–4]. In the realm of improving quality metrics, it is important to conduct research on developing and implementing effective metrics that take into account various aspects of quality, such as reliability, performance, and security [3]. In terms of addressing software security issues, it is necessary to analyze and improve methods for identifying and eliminating potential vulnerabilities and to develop tools that enhance compliance with security requirements in a growing environment of cyber threats [4–6]. Thus, recent studies require further investigation to continue comprehensive work on resolving the identified challenges.

Setting the task. Aim of the work is to study effective methods and tools for ensuring the quality of software systems and to develop a new comprehensive model for ensuring the quality of software systems, which combines the methods of module testing, integration testing and continuous integration, which contributes to the improvement of the quality of the software product at various stages of development.

Achieving the set goal comes down to solving the following problems:

- conducting a generalized analysis of current issues related to the study of effective methods and means to ensure the quality of software systems;
- conducting a comprehensive contextual base-review generalization of the main modern qualitative and quantitative methods that are used to assess the quality of software systems;
- analysis of the main mathematical models involved in integration testing and quality assessment of software systems;
- development and research of a new complex model of quality assurance of software systems, which is based on a combination of methods of module testing, integration testing and continuous integration.

THE MAIN PART

Conducting a generalized analysis of current issues related to the study of effective methods and tools for ensuring the quality of software systems, it is appropriate to highlight a number of issues in this area, which are presented in Table 1.

Table 1 is made by the author of this article on the basis of works analysis [1–15].

Table 1. Generalized analysis of current issues related to the study of effective methods and tools to ensure the quality of software systems

The name of the generalized area of topical issues	Generalized description of the problem	Research that is needed within the framework of solving this issue
Automated testing	Lack of full test coverage, which can lead to incorrect operation of the program in real conditions	Research within the development of automated tests that cover various aspects of the program and its functionality
Quality metrics	Lack of clear metrics for measuring software quality	Research within the development and implementation of effective metrics that take into account various aspects of quality such as reliability, performance and safety
Software security	Ensuring the security of software systems in the growing environment of cyber threats	Research that involves the analysis and improvement of methods for identifying and eliminating potential vulnerabilities, as well as the development of means to improve compliance with security requirements
Development methodologies	The choice of the most appropriate methodology for the development of software systems, taking into account the trends in ensuring the quality of the software product.	Research that involves a comparative analysis of various methodologies, identifying their advantages and disadvantages, as well as developing recommendations for choosing the appropriate approach to development
Quality standards	Lack of uniform quality standards for different fields of development.	Research on the development and implementation of quality standards that take into account the specifics of various types of software systems
Version control of software systems and configuration management	Effective implementation and management of changes in software code	Research on the development of tools for automated version control and configuration management to ensure code stability and quality

Shown in Table 1 the results of a generalized visual analysis show that the quality of software systems has several key aspects that need attention and research. In particular, in the field of resolving issues on automated testing, it is advisable to conduct research on improving software testing, which is used to prevent improper operation of programs and accordingly covers the analysis of their functionality in real conditions [1–4].

With the limits of improving quality metrics, it is important to conduct research on the development and implementation of effective metrics that take into account various aspects of quality, such as reliability, productivity and security of software [3]. In the framework of adaptive development, quality metrics in our opinion are advisable to apply the definition of key quality indicators (KPIs) to measure different aspects of software quality.

Fig. 1 shows a scheme of complex contextual basis-view generalization of basic modern qualitative methods that are used to assess the quality of software systems. Analyzing in Fig. 1. Methods should be noted that today, in the field of quality assessment of software systems, specific standards do not determine the

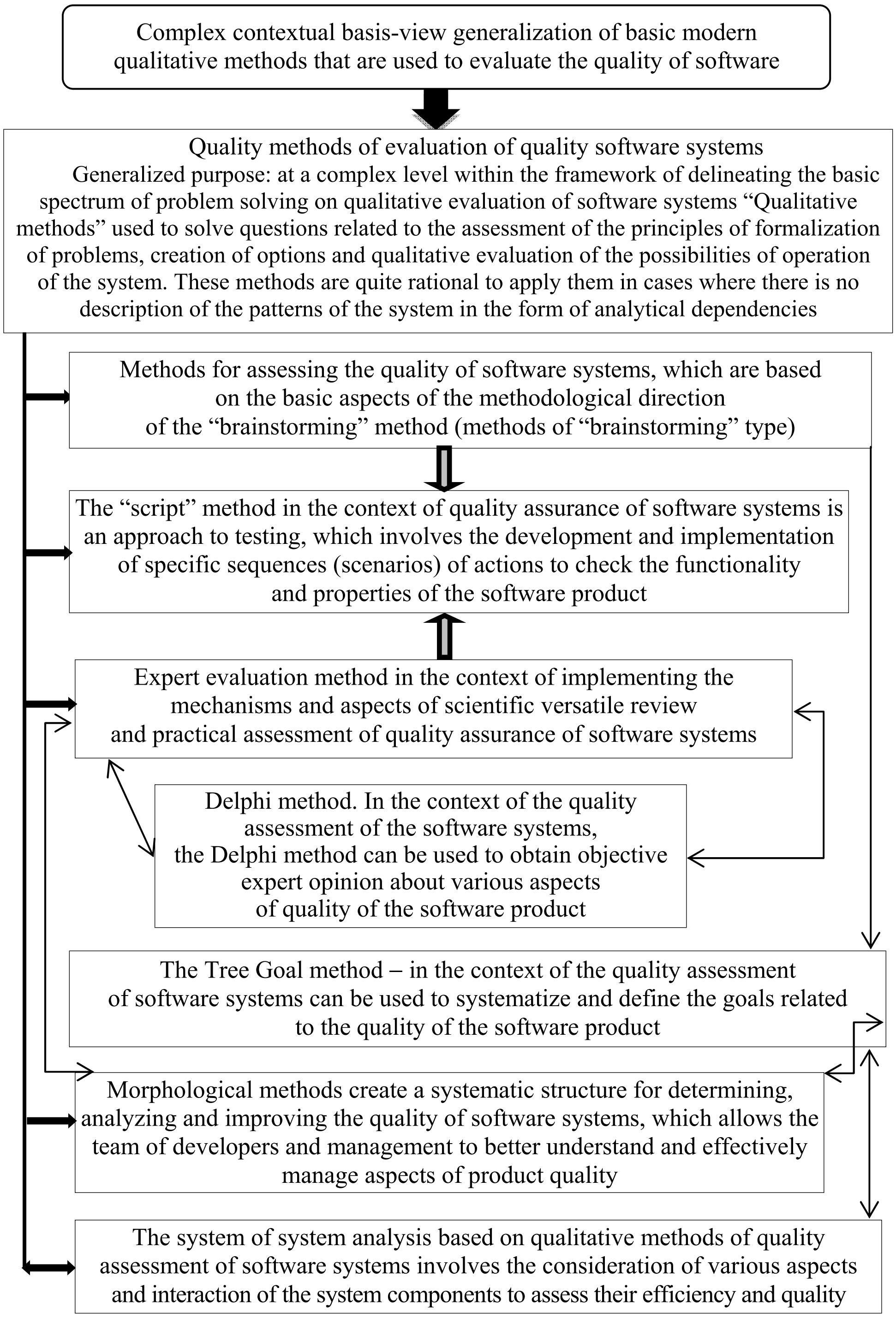

Fig. 1. Scheme of complex contextual basis-view generalization of basic modern qualitative methods that are used to assess the quality of software systems

use of specific brainstorming methods. However, brainstorming methods should be used in quality management processes and software testing, as well as in collecting requirements and design [9].

Brainstorming methods can be successfully integrated into these standardized processes to achieve better results in solving software quality problems [1–14]. Analyzing the script method in the context of quality assurance of software systems, we note that this is an approach to testing, which involves the development and implementation of specific sequences (scenarios) of actions to check the functionality and properties of the software [6]. Brainstorming methods can be used in the development of test scenarios, identification of potential defects and collecting requirements for testing.

According to ISO/IEC 9126 (Software Engineering – Product Quality): ISO/IEC 9126 is related to the quality of the software. Brainstorming methods can be used in determining the requirements for functionality, reliability, ease of use, etc.

Below are examples of how brainstorming can be used in the context of quality assessment of software systems:

- ISO/IEC 25010 (Square): ISO/IEC 25010 Standard "Systems and Software. Model of Quality and Assessment" determines the software quality model. Brainstorming methods can be used in determining the requirements, as well as in the analysis and planning of testing to ensure the completeness and variety of test scenarios;
- IEEE 730 (Standard for Software Quality Assurance Processes): The IEEE 730 Standard defines the quality assurance processes. IstQB (International Software Testing Qualifications Board): ISTQB initiative defines standards for certification of software testing professionals.

It is worth noting that the standards do not fully specify the fulfillment of the completeness of the use of brainstorming methods, they provide context and principles for the introduction of creative and collective approaches to solving problems in the quality of software systems.

In Table 2 the generalized characteristics of methodological directions of qualitative methods in the spectrum of quality of software systems are given. Brainstorming methods can be successfully integrated into these standardized processes to achieve better results in solving software quality problems [1–14].

Analyzing the script method in the context of quality assurance of software systems, we note that this is an approach to testing, which involves the development and implementation of specific sequences (scenarios) of actions to check the functionality and properties of the software [6]. Brainstorming methods can be used in the development of test scenarios, identification of potential defects and collecting requirements for testing.

According to ISO/IEC 9126 (Software Engineering —Product Quality): ISO/IEC 9126 is related to the quality of the software. Brainstorming methods can be used in determining the requirements for functionality, reliability, ease of use, etc.

Below are examples of how brainstorming can be used in the context of quality assessment of software systems:

- ISO/IEC 25010 (Square): ISO/IEC 25010 Standard "Systems and Software. Model of Quality and Assessment" determines the software quality model. Brainstorming methods can be used in determining the requirements, as well as in the analysis and planning of testing to ensure the completeness and variety of test scenarios;
- IEEE 730 (Standard for Software Quality Assurance Processes): the IEEE 730 Standard defines the quality assurance processes. IstQB (International Soft-

ware Testing Qualifications Board): ISTQB initiative defines standards for certification of software testing professionals.

It is worth noting that the standards do not fully specify the fulfillment of the completeness of the use of brainstorming methods, they provide context and principles for the introduction of creative and collective approaches to solving problems in the quality of software systems.

In Table 2 the generalized characteristics of methodological directions of qualitative methods in the spectrum of quality of software systems are given.

Quantitative methods used to evaluate the quality of software systems have the following basic generalized characteristics:

- quantitative formalization of problems: quantitative methods allow to structure and formalize problems using quantitative indicators and parameters. Provides objectivity and quantitative evaluation of system parameters;

Table 2. Generalized characteristics of methodological directions of qualitative methods in the spectrum of quality assessment of software systems

The name of the methodological direction	Description	Advantages and disadvantages
Testing	It includes the execution of programs in order to detect errors or deficiencies	Effectively detects some types of errors, helps to confirm compliance with requirements. Cannot guarantee the absence of all errors, testing costs may be high
Code analysis	Expert analysis of software code to identify potential problems or inefficiencies	Can reveal code complexity, potential vulnerabilities. Limited to the quality of the code, it is difficult to detect certain types of errors
Automated quality analysis tools	Using tools for automated code quality analysis and error detection	Reduces dependence on the human factor, quickly detects common problems. May underestimate context and complex aspects of code
Code check	Code review by the development team to identify errors and improve quality	Attracts experts, promotes knowledge exchange, is effective in identifying shortcomings. Requires time and effort from the development team
Quality modeling	Use of mathematical models	Allows to carry out quality analysis before product release

Specified in Table 2 methodological directions can be used separately, or in combination to achieve a more accurate and complete assessment of the quality of software systems.

In Fig. 2 shows a scheme of complex contextual basis-view generalization of basic modern quantitative methods that are used to assess the quality of software systems. Accordingly [3–4] quantitative methods within the framework of the implementation of the quality of software systems allow quantitative analysis and make sound solutions based on numerical data and system parameters:

- quantitative formation of variants: quantitative methods help to create different variants of system implementation, taking into account quantitative aspects. Provides objectivity and quantitative evaluation of the possibilities of different options;

- quantitative evaluation of operation options: used to quantify the efficiency and suitability of different options for the system. Allows objective analysis and comparison of different scenarios of system use;

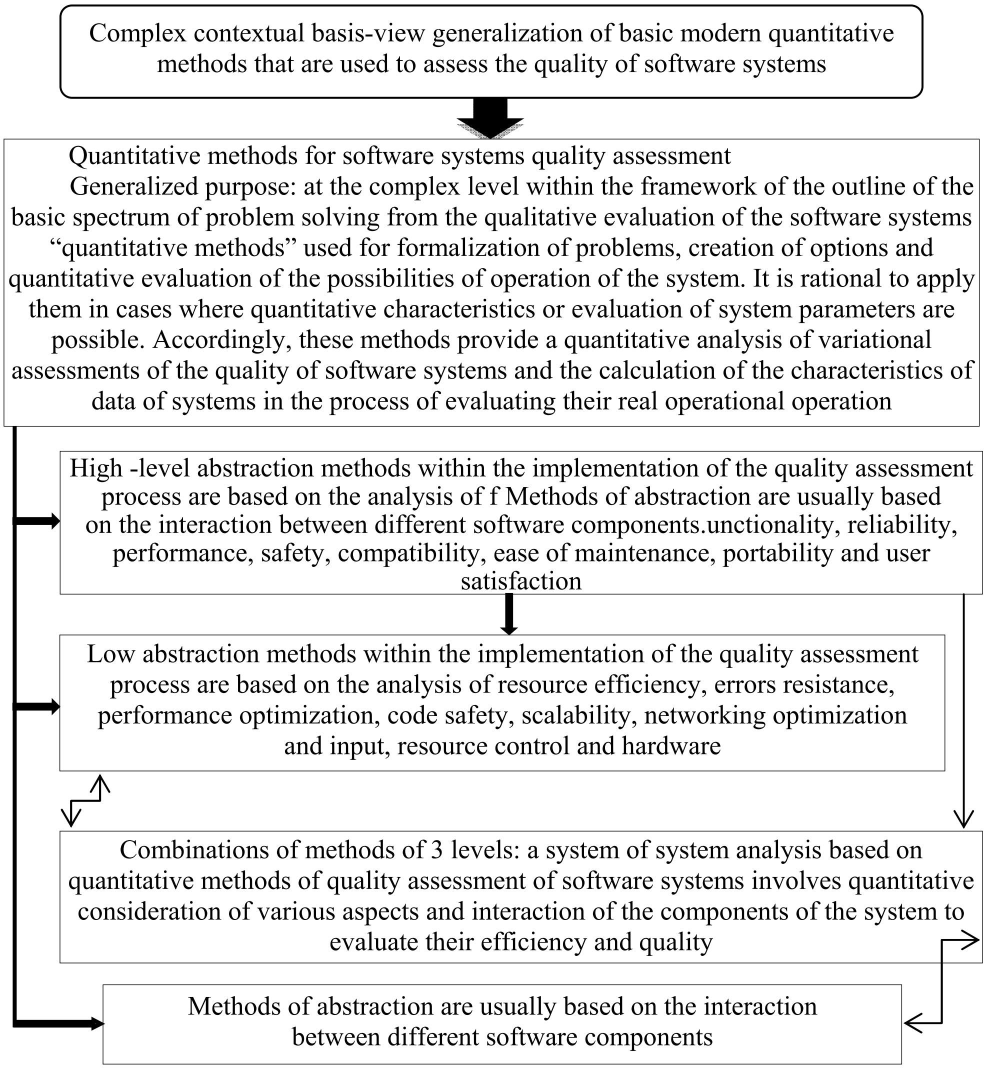

Fig. 2. Scheme of complex contextual basis-view generalization of the main modern quantitative methods that are used to evaluate the quality of software systems

- application in the presence of analytical dependencies: quantitative methods are effectively used when the possibility of expression of the system analytically allows quantitative characteristics. Provides accuracy and objectivity in the presence of quantitative data.

The modular testing method is closely linked to the concept of software testing and arose in the context of software testing. However, the specific term "modular testing" and its methodology received significant flowering through a movement known as "Extreme Programming" (XP), which began in 1990–2000 (developers: Kent Beck and his colleagues). XP is a software development methodology that defines certain practices and principles to improve the quality and speed of development. It should be noted that modular testing involves writing automated tests for each individual "module" of the code before its integration into the system. The essence of this method is to create the maximum number of automated tests for the individual components of the program and run them during development to detect errors in the early stages.

Integration testing is an important component of the quality assessment of software systems and is usually performed at the development stage when the different components of the software system are grouped into a single system. The main purpose of integration testing is to check the interaction between these components, as well as integrated system functions.

Different approaches and techniques are used in the integration testing process, such as:

- testing of modules on isolation: before integration of modules into the system, each module is tested separately to check its functionality and compliance with requirements;
- interface testing: an important step in integration testing is to check the interaction between components through their interfaces. This may include data testing between modules and verifying the data of data integrity;
- testing of the interaction of components: after the integration of the modules test the interaction between them, convincing that they work properly together and perform the expected functions;
- exception and error testing: integration testing also includes checking the system response to exceptional situations and errors, such as incorrect data or inaccessible resources;
- productivity testing: during integration testing, performance tests are also performed to ensure that the system works efficiently and is capable of processing the load caused.

The purpose of integration testing is to ensure the high quality of the software system by identifying and solving problems related to the integration of components. This allows you to ensure the correct and reliable operation of the system as a whole.

In Table 3 the results of the analysis of the main mathematical models involved in integration testing and quality assessment of software systems are presented. From Table 3 it is clear that each model has its own unique essence and purpose, which affects the process of evaluating the quality of software systems. The interaction matrix and interaction graphs allow the visualization of the structure and interaction between the components of the software system, which contributes to the identification of weaknesses and the analysis of the complexity of interaction. Models of interaction, in turn, enable a mathematical description of this interaction, which allows to carry out a deeper analysis and forecasting of the behavior of the system.

It should also be noted that when choosing models to evaluate the quality of software systems, it is important to consider their advantages and disadvantages, as well as the context of the project and the specifics of the system. The combination of different models can be useful to obtain a more complete view of the system's state and test efficiency.

The constant integration method (CI) is a key practice in the field of software development aimed at improving the quality of software systems. The main idea of the CI is to regularly and automatically combine code changes in a common repository and perform automatic tests to check the correctness of these changes. Below is a description of the constant integration method and its impact on the quality assessment of software systems:

- code changes automated: developers regularly make changes to the project code. Thanks to the CI, these changes automatically merge (integrate) into the main branch of the repository. This allows you to identify conflicts and other problems before they get into the production environment;

- automatic tests: after each change of change, the CI system automatically starts tests that check the functionality, performance, safety and other aspects of the software system. This allows you to identify possible problems during the early stages of development;

Table 3. Analysis of basic mathematical models involved in integration testing and quality assessment of software systems

Mathematical model	Essence	Appointment	Advantages	Disadvantages
Interaction matrix	Representation of interaction between components in the form of a matrix	Visualization and analysis of the interaction structure, identification of “weak places”	Ease of use, quick identification of defects	Does not take into account the dynamics of interaction, limited in use for complex systems
Interaction graph	Representation of interaction between components in the form of a graph	Visualization and analysis of complex interaction, detection of cycles and ways of interaction	Detailed overview of the interaction structure, help with architecture analysis	Difficult to use for large systems
Interaction models	Mathematical models describing the interaction between components	Analysis and prediction of interaction between components, identification and elimination of shortcomings	Deep interaction analysis, possibility of predicting system behavior	Requires a large amount of data and computing resources, difficult to implement
Probability models	Mathematical models that determine the probability of various events	Assessment of risks and effectiveness of testing, detection and elimination of shortcomings	Objective assessment of efficiency and risks, the possibility of making decisions based on probability	It is difficult to model all possible scenarios and factors affecting the probability
Testing models	Models describing the process of testing a software system	Organization of testing, determination of the most effective approaches and strategies	Systematization and structuring of the testing process, the possibility of choosing the best approaches	Requires precise definition of testing parameters and criteria, may not be flexible enough in some situations

- constant feedback: if problems are identified during the performance of tests, the developers receive notifications about it, which allows them to quickly correct mistakes and improve the quality of the code;
- automated deployment: some CI systems may automatically deploy changes to the test or production environment after successful completion of the tests. This allows you to quickly and effectively introduce new functionality and correction of errors;
- code quality metrics: some CI systems may include code analyzers and quality metrics that automatically estimate the level of readability, efficiency and other aspects of code quality.

The constant integration method allows you to create higher quality software systems, reducing the time of detection and correcting errors, facilitating the joint work of developers and increasing the reliability of software as a whole.

MODELING AND TESTING

In the range of modeling of a new complex model of quality assurance of software systems, based on the combination of modular testing, integration testing and constant integration methods, it is important to keep in mind a number of initial assumptions. In particular, given the automation of processes in our case, it is suggested that many stages of testing and integration can be automated. This means that the creation and performance of tests can be carried out without significant intervention.

Given the specifics of the modular testing, it is suggested that the individual components of the program (modules) are tested in isolation from other components to ensure that they work properly isolated from other parts of the program.

Given the specifics of integration testing, it is suggested that after successful modular testing, the components of the program are gradually combined and tested as a holistic system to believe that they are cooperating correctly.

Continuous delivery involves the assumption that the processes of correction of errors, testing and release of software are automatically and continuously to ensure rapid delivery of changes and updates.

Considering the needs of assembling the assembly and deployment at the complex level, it is suggested that the processes of collecting the software code and its automatic deployment into the test environment or the productive server must also be automated.

Considering the need for automation tools, there is an assumption that there are appropriate tools and technologies for the implementation of automated testing, integration and constant integration processes. Such tools can include media for testing, version control systems, automated testing tools and more. With the above initial assumptions, you can start modeling and implementation of a new complex model of quality assurance of software systems.

Mathematical apparatus for the developed new model of complex quality assurance of software systems, based on the combination of modular testing methods, integration testing and constant integration can be presented as follows.

1. Tasks of the modular testing procedure of the software system:

$$S_u = N_u \times P_u,$$

where S_u — the overall success of modular testing; N_u — the total number of wise tests; P_u — predicted probability of successful passage of one modular test.

2. Tasks of the procedure testing procedure:

$$S_i = N_i \times P_s,$$

where S_i — the overall success of integration tests; N_i — the total number of modular tests; P_i — The probability of successful passage of one integration test.

3. Tasks of the continuous integration procedure:

$$C = F_C \times T_C,$$

where C — total number of committees (variable code); F_C — frequency of committees; T_C — total time during which committees were carried out.

4. Tasks of the procedure of continuous integration and constant delivery:

$$R = C / T_C$$

where C — total number of committees (variable code); R — the speed of releases, which is determined by the number of committees and time; T_C — the total time during which committees were carried out.

5. The task of determining the quality of the software system:

$$Q = S_u \times S_i \times R,$$

where Q — The total quality of the software system.

Subsequently optimizing the mathematical apparatus for the model of complex quality assurance of software systems, which is based on the combination of modular testing methods, integration testing and constant integration, may include the following characteristics and methods:

Code quality metrics: used to quantify the quality of the software code. These metrics may include code coating tests, defects, test time, and more.

Statistical methods: used to analyze the results of errors, test efficiency and system stability.

Probability and statistics theory: used to calculate the likelihood of defects during testing, risk assessment and statistical significance of test results.

Optimization methods: used to increase the efficiency of testing and select the optimal test strategies taking into account resource restrictions.

Machine learning and artificial intelligence algorithms: used to automate the test analysis processes, identify anomalies and forecast possible problems.

These methods and tools allow you to create a complete and optimized complex mathematical model to ensure the quality of software systems, which takes into account the interaction of different test methods and their impact on the quality and reliability of the software.

Description of the algorithm of practical implementation of the developed model. In practice, the implementation of the developed complex model of quality assurance of software systems will include several steps and stages:

1. Planning and setting up the environment:

- Setting up the version control system (GIT example) for constant integration.
- Installation and adjustment of tools for automated testing (for the example of JUnit for modular testing, Selenium for automated integration testing).

2. Creating tests:

- Developing a set of modular tests for each component of the program.
- Writing integration tests to check the interaction between components.

3. Setting up constant integration (PI):

- Create configuration files for automatic assembly and testing after each committee.
- Setting up integration with the task management system or notification of the developers about the results of the testing.

4. Performing tests and analysis of results:

- Automatic performance of modular tests and integration tests after each committee.
- Analysis of test results and detection of errors or disadvantages in the code.

5. Control of versions and releases:

- Management of software versions through the version control system.
- Automatic software release based on test results and successful integration.

6. Monitoring and reporting:

- Monitoring the work of the continuous integration system and test results.
- Generation of code quality reports, testing and testing results for analysis and improvement of the development process. This process requires the work of developers, testers and operators, as well as the introduction of a number of tools for automation of routine testing and integration tasks.

The developed model of quality assurance of software systems, which combines modular testing, integration testing and constant integration, has several useful advantages:

Improved software quality: this model allows you to identify and correct errors in the early stages of development, providing high quality software.

Reducing the time for detecting and correcting errors: constant integration and continuous delivery allow you to identify and correct problems quickly, which reduces the time of development and improves the speed of product production.

Testing and release automation: through automated software testing and release processes, the risks of human errors can be reduced and the project stability can be reduced. Improving development efficiency: developers can focus on writing code, as many routine tasks (testing, assembly, deployment) are automated. Reduced testing costs and releases: automated quality assurance processes save time and money that is usually spent on manual testing and releases.

Increasing user confidence: high product quality and renewal speed helps to improve users' reputation and trust.

In general, this model helps to make the software development process more efficient, faster and reliable, which is critical in the modern fleeting world of software development.

In Table 4 the results of practical testing of plowing methods and the developed model in the analy sis of the quality of the application "task management system".

Table 4. Analysis of comparison of software testing methods in the analysis of the quality of the application "task management system"

Features	Modular testing	Integration testing	Continuous integration	Combination of methods (unit testing + Integration + PI)
Execution time	5 hours	8 hours	12 hours	10 hours
Number of detected errors	10	50	30	70
Level of automation	High	Average	High	High
Stability of releases	High	average	High	High
Test coverage (percentage)	90%	70%	95%	97%
Stability (scale 1-10)	9	7	8	8

From Table 4 it is observed that software quality testing by the developed method takes 10 hours, which is less than continuous integration and close to the average time required for integration testing. Thus, in terms of execution time, the combined method is quite efficient. Analyzing the number of errors detected: the combined method detects 70 errors, which is more than any other testing method. This indicates its high efficiency in detecting errors in software. Analyzing the level of automation: the combined method has a high level of automation, which allows for efficient testing without significant human involvement. Analyzing release stability and stability: the combined method has a high level of release stability, which is equal to the release stability obtained with continuous integration. This is important to ensure software quality. Analyzing test coverage: the combined method has the highest rate of test coverage — 97%. This indicates that it tests more parts of the software and detects more possible problems. So, from the standpoint of comparison with existing testing methods, the combined method (Unit Testing + Integration + PI) is very effective, as it combines the advantages of different approaches and ensures high quality and stability of the software. In turn, it should be noted that in order to obtain the results of practical testing of the described methods and the developed quality model during the quality analysis of the "task management system" application, various tools were used, including: Automated test frameworks: popular test frameworks were used for module testing and integration testing, such as like JUnit, NUnit, or PyTest. The above-mentioned test frameworks allow you to automate the execution of tests and analyze their results.

Systems for continuous integration: tools such as Jenkins, Travis CI, or GitLab CI were used for continuous integration. These systems allow you to automate the process of building, testing, and deploying software with each code change.

Code quality monitoring and analysis: code quality monitoring tools such as SonarQube or CodeClimate were used. These tools allow you to identify potential problems in your code, such as code duplication, improper use of variables, and other anomalies. Resource Usage Analyzers: profilers and performance analyzers such as VisualVM have been used to analyze the application's resource usage, and YourKit is also possible. These tools allow you to identify performance and memory usage issues in an application. Bug trackers: used bug trackers like Jira, Bugzilla, GitHub Issues to track the issues found and fix them later. These tools contributed to obtaining practical results of testing and quality analysis of the "task management system" application using the described testing methods and the developed quality assurance model. The application of this strategy and approaches is quite expedient in terms of improving configuration management processes and identifying vulnerabilities in software systems.

CONCLUSIONS

The analysis of the current state of the issue raised in the article showed that an important task is to further improve the methods and means of quality assurance of software systems in the context of rapid technology development. Research confirms that effective methods and quality assurances are a key element of successful development of software systems. In the course of modeling, a new complex model of quality assurance of software systems was developed, which is based on the combination of modular testing methods, integration testing and constant integration. The advantage of this development is to increase its adaptation to solving the main tasks for the quality of software systems. The results of practical research have shown that the integrated use of methods, such as modular testing, integration testing and continuous integration, better facilitates the detection and correction of errors in the early stages of software development. The prospects for further research are to improve existing techniques, as well as to study the latest technologies that can help improve the quality of software in the future.

REFERENCES

1. H. Foidl, M. Felderer, "Integrating software quality models into risk-based testing," *Software Quality Journal*, vol. 26, no. 2, pp. 809–847, 2016. doi: https://doi.org/10.1007/s11219-016-9345-3
2. K. Sahu, R.K. Srivastava, "Predicting software bugs of newly and large datasets through a unified neuro-fuzzy approach: reliability perspective," *Advances in Mathematics: Scientific Journal*, vol. 10, no. 1, pp. 543–555, 2021. doi: https://doi.org/10.37418/amsj.10.1.54
3. K. Sahu, Fahad A. Alzahrani, R.K. Srivastava, R. Kumar, "Evaluating the Impact of Prediction Techniques: Software Reliability Perspective," *Computers, Materials & Continua*, vol. 67, no. 2, pp. 1471–1488, 2021. doi: https://doi.org/10.32604/ cmc.2021.014868
4. S. Sackey, D.-E. Lee, B.-S. Kim, "Duration Estimate at Completion: Improving Earned Value Management Forecasting Accuracy," *KSCE Journal of Civil Engineering*, vol. 24, issue 3, pp. 693–702, 2020. doi: https://doi.org/10.1007/s12205-020-0407-5
5. P. Sharma, A.L. Sangal, "Building and Testing a Fuzzy Linguistic Assessment Framework for Defect Prediction in ASD Environment Using Process-Based Software Metrics," *Arabian Journal for Science and Engineering*, vol. 45, issue 12, pp. 10327–10351, 2020. doi: https://doi.org/10.1007/s13369-020-04701-5

6. Y. Hassouneh et al., “Boosted Whale Optimization Algorithm with Natural Selection Operators for Software Fault Prediction,” *IEEE Access*, vol. 9, pp. 14239–14258, 2021. doi: https://doi.org/10.1109/access.2021.3052149
7. Rafa Al-Qutaish, “Quality Models in Software Engineering Literature: An Analytical and Comparative Study,” *Journal of American Science*, vol. 6, pp. 166–175, 2010.
8. J. Estdale, E. Georgiadou, “Applying the ISO/IEC 25010 Quality Models to Software Product,” in Larrucea X., Santamaria I., O’Connor R., Messnarz R. (eds) Systems, Software and Services Process Improvement. EuroSPI 2018. Communications in Computer and Information Science, vol. 896, 12 p., 2018. Springer, Cham. doi: https://doi.org/10.1007/978-3-319-97925-0_42
9. C. Zhang B. Li, L. Wang, H. Xu, T. Shao, “A Hierarchical Model for Quality Evaluation of Mixed Source Software Based on ISO/IEC 25010,” *International Journal of Software Engineering and Knowledge Engineering*, 33(02), pp. 1–25, 2022. doi: https://doi.org/10.1142/S021819402250070X
10. Elder Bruno Evaristo Correa, Jeffson Celeiro Sousa, Antônio Jorge Gomes Abelém, Sandro Ronaldo Bezerra Oliveira, “An evaluation of Security Features based on Iso/Iec 25023 for a Distributed Autonomic Scientific Publisher Tool on a Permissioned Blockchain,” *Journal of Information Systems and Technology Management*, vol. 19, 2022. doi: https://doi.org/10.4301/S1807-1775202219020
11. J.-X. Chen, “Overall performance evaluation: new bounded DEA models against unreachability of efficiency,” *Journal of the Operational Research Society*, vol. 65, issue 7, pp. 1120–1132, 2014. doi: https://doi.org/10.1057/jors.2013.57
12. M.-A. Filz, C. Herrmann, S. Thiede, “Simulation-based Assessment of Quality Inspection Strategies on Manufacturing Systems,” *Procedia CIRP*, vol. 93, pp. 777–782, 2020. doi: https://doi.org/10.1016/j.procir.2020.04.069
13. A. Golabchi, S. Han, S. AbouRizk, “A simulation and visualization-based framework of labor efficiency and safety analysis for prevention through design and planning,” *Automation in Construction*, vol. 96, pp. 310–323, 2018. doi: https: //doi.org/10.1016/j.autcon.2018.10.001
14. P. Han, L. Wang, P.X.K. Song, “Doubly robust and locally efficient estimation with missing outcomes,” *Statistica Sinica*, 26(2), 2016. doi: https://doi.org/ 10.5705/ss.2014.030
15. L. Hund et al., “Distinguishing between model- and data-driven inferences for high reliability statistical predictions,” *Reliability Engineering & System Safety*, vol. 180, pp. 201–210, 2018. doi: https://doi.org/10.1016/j.ress.2018.07.017

Received 04.04.2024

INFORMATION ON THE ARTICLE

A.S. Shantyr, ORCID: 0000-0002-0466-3659, State University of Information and Communication Technologies, Ukraine, e-mail: shantyr.a@duikt.edu.ua

ВИКОРИСТАННЯ МЕТОДІВ ТА ІНСТРУМЕНТІВ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНИХ СИСТЕМ / А.С. Шантир

Анотація. Запропоновано розгляд ефективних методів та засобів для забезпечення якості програмних систем. У спектрі даної тематики розглянуто актуальні питання, пов’язані із забезпеченням якості програмних систем у межах реалізації аналізу методів та засобів, які застосовуються на практиці для забезпечення розроблення високоякісного програмного забезпечення. У ході моделювання розроблено нову комплексну модель забезпечення якості програмних систем, яка ґрунтується на поєднанні методів модульного тестування, інтеграційного тестування та постійного інтегрування. Перевагою цієї розробки є підвищення її адаптування до вирішення основних завдань із забезпечення якості програмних систем. На базі розробленої моделі запропоновано стратегії та підходи щодо вдосконалення процесів управління конфігурацією та виявлення вразливостей програмних систем.

Ключові слова: комплексна модель, стандарти якості, інтеграційне тестування, модульне тестування, технологічні виклики, інтеграційне тестування, неперервне інтегрування.