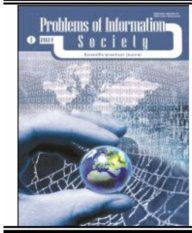

www.jpis.az

15 (1)
2024

Developing a conceptual model for improving the software system reliability

Tamilla A. Bayramova[1], Nazakat C. Malikova[2]

[1,2] Institute of Information Technology, B. Vahabzade str., 9A, AZ1141 Baku, Azerbaijan

tamilla@iit.science.az[1], naranara_68@mail.ru[2]

orcid.org/0000-0002-8377-3572, *orcid.org/0000-0001-9617-0554*

ARTICLE INFO

http://doi.org/10.25045/jpis.v15.i1.05

Article history:
Received 11 September 2023
Received in revised form
13 November 2023
Accepted 15 January 2024

Keywords:

Software Quality
Software reliability
Quality model
Classification of defects
Testing
Reliability model
Reliability parameters

ABSTRACT

In the Industry 4.0 environment, the software systems development methodology is rapidly evolving, flexible technologies and new programming languages are being applied. The development of the software industry has made the issue of the quality of software systems an urgent problem. A number of quality models have been proposed for determining the quality of software systems so far, and these models specify the parameters and criteria for evaluating quality. Software reliability is one of the key indicators among the quality parameters of software systems, as it quantifies software crashes which can bring down even the most powerful system, ensuring that software systems run correctly and unexpected incidents do not occur. The increasing difficulty of the software system, the expansion of the scope of issues assigned on them, and as a result, the significant increase in the volume and complexity of the software system have made the problem of the reliability of the software system even more urgent. The essence of the issue is to reveal the main factors affecting the reliability of software systems, demonstrate existing problems in this area and develop mathematical models for assessing reliability. Mathematical models estimate the number of errors remaining in the software system before commissioning, predict the time of occurrence of the next crash and when the testing process will end. It is necessary to comprehensively approach the issue of ensuring reliability at all stages of the life cycle of the software system. This paper proposes a conceptual model to solve this problem.

1. Introduction

Information technology is one of the intensively developing main areas of modern life. Complex information systems are used to solve large-scale scientific and industrial problems. They include complex issues such as artificial intelligence and management of special purpose objects, starting with simple issues such as collecting, processing and protecting information. Software systems control nuclear reactors, and are applied in aviation, medicine, banking and other areas of human activity. The creation of cyber-physical systems enable real-time systems to share data instantaneously and freely and to make free decisions, and robotics provides significant support in highly dangerous environments for humans.

The introduction of automated information systems, along with reducing the number of operations performed by humans, has also created new problems. As a result of serious errors in software systems, man-made disasters occur, companies suffer from huge damages, large-scale accidents happen in the work of infrastructure networks, and people lose their lives. The emergence and development of cyber-physical, cyber-biological systems, intelligent sensor networks, artificial intelligence, nano and supercomputer technologies, robotics, and unmanned aerial vehicles have exacerbated security-related problems and added new ones (Alguliyev et al., 2019). Providing information security of cyber-physical systems and preventing cyber-attacks has become one of the challenging problems to solve (Bayramova, 2022).

Current software industry must produce high-quality software systems to succeed in a competitive market. Therefore, many companies spend a lot of money on modernizing their software development technologies. Currently, some enterprises have accumulated enough experience using metrics to manage the quality of software products developed and implemented. Moreover, successful projects individually develop important metrics based on the characteristics of the work and the characteristics of the application area.

Software quality is a complex and multifaceted concept. There are different approaches to quality in software engineering. The quality of the software product refers to 1) the compliance of its characteristics with the requirements specified, 2) the absence of errors and mistakes in the product, 3) the possibility of modifications in the software code and adding new functions. Software code quality management is one of the important issues of software engineering, and the development of quality software systems still remains one of the most pressing problems today.

Measuring these characteristics allows to check how well the software system meets the requirements and determine the time to finish software system testing, thereby saving time and budget.

The work aims to determine the software system reliability indicators and the main factors affecting its reliability of the based on the quality models, and to identify the problems in this field.

The paper proposes a conceptual model for software reliability assessment.

2. Related Work

Brereton et al. analyze published research on mentioned topics in software engineering. A systematic review of the literature based on empirical research in software development is summarized, a number of analyzes conducted by the authors and others are explained, and some recommendations are made for the application of this practice to software engineering (Brereton et al, 2007).

Lanxin Yang et al conducted a systematic review of 241 software quality literature published between 2004 and 2018. The results showed that in recent years, the goals in this field have become more concise, the tools more diverse and rigorous, and the quality criteria more considerate (Yang et al., 2021).

Sahu and Srivastava (2020) show that in order to improve the reliability of software systems, professionals attempt to measure, manage and predict reliability. They consider reliability growth models and their application at different stages of software system development. Studies show that there is no general forecasting model that covers all processes. The authors propose the development of a generalized model applicable at all stages to predict the software system reliability. Such a model will reduce the time and cost of applying several tools to predict reliability at different stages (Sahu & Srivastava, 2020).

Cristescu et al. describe the principles and methods that form the basis of software reliability assessment, starting with the definition of concepts that express the characteristics of software quality. The reliability assessment aims to analyze the risk and reliability of software systems (Cristescu et al, 2015).

Maevsky et al (2017) analyze and classify software reliability publications over the past 50 years. The classification is built using a facet-hierarchical approach, which allows to systematize scientific publications in the field of software reliability. Based on statistics and the number of publications according to different classification criteria, a retrospective analysis of changes in this topic is conducted (Maevsky et al, 2017).

Felipe Febrero et al. point out that there is a real need to develop effective methods for creating reliable software systems. To this end, they analyze international quality standards. They emphasize that it is important to develop the proposed SRGM models according to these standards to ensure that they are not only reliable, but also well-founded and cost-effective to implement (Febrero, Calero, & Moraga, 2016).

Yadav H. and Yadav D. point out that analyzing and assessing the reliability of software systems early in the software life cycle leads to improved quality while reducing the time and cost of testing. They propose a model to predict the density of software errors using fuzzy logic and reliability metrics collected in the early stages of the software life cycle. The proposed model was applied to twenty real projects. Experiments show that this indicator is higher in the requirements analysis phase than in the design and coding phase (Yadav & Yadav, 2017).

Alaswad and Poovammal review the application of machine learning algorithms in predicting the quality of software systems and the metrics and methods to be used. First, they give the quality indicators, which are the input data of the machine learning algorithms. After that, they analyze forecasting methods and mechanisms. The authors emphasiz the presence of professional experts as an important attribute in the development of quality software. They show that machine learning algorithms can provide more accurate predictions by being trained on pre-detected errors in the software (Alaswad & Poovammal, 2022).

Sahu and Srivastava highlight that numerical evaluation of reliability based on the detection of bugs, errors and failures leads to the construction of a more reliable system (Sahu & Srivastava, 2019).

Abdallah et al. propose a model to measure the quality of IoT systems. The authors emphasize that the Internet of Things affects all areas of life. Therefore, they must be of good quality. They use reliability and scalability as the main characteristics (Abdallah et al., 2019).

Jalilian and Mahmudova propose a new method for error detection using the Imperialist Competitive Algorithm in their article.

3. About the quality of the software system

Low quality in real-time systems, management, aviation, medicine and other critical systems can lead to economic losses, environmental problems, loss of human life, etc. Following the Mariner 1 control system accident that resulted in its loss, the US Air Force Board decided to conduct code expertise (the analysis of the software code by another expert after the software was developed) in the software development process for the first time in 1962 (Daniels et al, 2003).

Systematic research on the quality of software systems has been started since the 70s of the 20th century, and even now many researchers focus on this problem (Musa & Everett, 1990; Al-Qutaish, 2010; Ndukwe et al, 2023). The Consortium for Information and Software Quality published a study entitled "The Cost of Poor Software Quality in the US: A 2022 Report" (2023). This report quantifies the impact of low-quality software on the US economy. In 2022, the damage caused by such programs was estimated at 2.41 trillion USD. Of this, 1.52 trillion USD is accounted for by software crashes. The report focuses on the following areas to address software deficiencies:

- Quality standards / taxonomy of software problems;
- Tools for understanding, searching and eliminating deficiencies;
- Artificial intelligence / machine learning tools.

The standards of software engineering are developed, improved and expanded as a result of the integration, regulation and optimization of advanced methods and theories developed as a result of research and accepted in enterprises. The role of international standards in the evolution of software systems is great. Various standardization organizations (ISO/IEC, ANCI, IEEE) have developed hundreds of international standards to improve software quality. These standards are developed based on research results and incorporate advanced methods and theories. They support and regulate the software life cycle processes, and at the same time, they are applied to reduce the costs incurred in the production of software systems, to control the quality of software products and to increase the quality of services related to them (Bayramova, 2020).

Different definitions of software quality have been proposed by standardization organizations. Software Quality:

- is a set of features and characteristics that ensure the ability of a product or service to meet specified or intended requirements (ISO 9001).
- is the degree to which software has a desired combination of attributes (IEEE 1061).
- is the planned and systematic arrangement of all activities necessary to reasonably assure that an item or product conforms to specified technical requirements (IEEE 730).

If we summarize them, we can conclude that the quality of the software system is the degree of compliance of the functional, technical, operational characteristics of the ready-to-use software tools with the goals and requirements set before their development.

The concepts of software product quality and software code quality are distinguished. The quality of a software product is determined by how well it meets the requirements of users. If at the first stage the analysts are not able to correctly identify the issues and problems of the users, then the software product that does not meet the requirements cannot be called quality.

The quality of the software code refers to its properly designed architecture, strict structuring, precise division of the code into functional blocks, placement of components, correct design of connections between them, absence of errors, etc. Standards, various methodologies and practices, and design templates are used to develop high-quality software code.

There is a fundamental difference between software system quality and software code quality. A software system may have quality written code; however, it is considered poor quality if it does not meet the requirements. Conversely, a software system may meet all requirements but have poor quality software code. These issues are checked during software system verification and validation. Consequently, quality software refers to software that has successfully passed both the validation and verification process (Bayramova & Abbasova, 2016).

4. Quality models of software systems

A number of quality models have been proposed for developing quality software systems and measuring quality so far (Al-Qutaish, 2010; Kumar & Gupta, 2017; Miguel et al, 2014). Quality models of software systems have been applied since the 70s of the 20th century and have maintained their importance in improving the quality of software systems. The first quality models were developed by McCall and Boehm (McCall et al, 1977; Boehm et al, 1978). Later, IEEE 1219, ISO 9126 and ISO 25010 models were developed by standardization organizations. Software systems quality models can be adapted to all areas (the Internet of Things, mobile applications, etc.) depending on the context. The most commonly used models ISO/IEC 9126 (2001) (Fig. 1.) and ISO/IEC 25010 (2011) (Fig. 2.) as its continuation are developed. The latter was revised and approved in 2017 and remains current.

The ISO 9126 quality model was developed based on McCall and Boehm's quality models. In this model, internal and external characteristics of software systems are defined and functionality metrics are added. The ISO 25010 model is an updated version of the ISO 9126 model, adding new characteristics such as security and compatibility (Gordieiev et al, 2014; Albeanu et al, 2020). This quality model is based on 8 main characteristics and their sub-characteristics (Jharko, 2021).

Fig. 1 and Fig. 2 show that the quality of a software system includes the main parameters such as functionality, usability, efficiency, supportability, portability, compatibility and security, and these, in turn, are grouped into criteria. These parameters and criteria provide consistent processes and terminology for defining, measuring, and evaluating system and software product quality.

It should be noted that all parameters of quality depend on each other. Although all parameters of quality are interdependent, one of them may be more important than the other. In general, if high values are obtained for all parameters, it is possible to guarantee the quality of the software system.

Thus, the main requirements for the quality of modern software:

- high reliability;
- protecting the completeness, correctness and accuracy of information;
- providing protection against unauthorized data access;

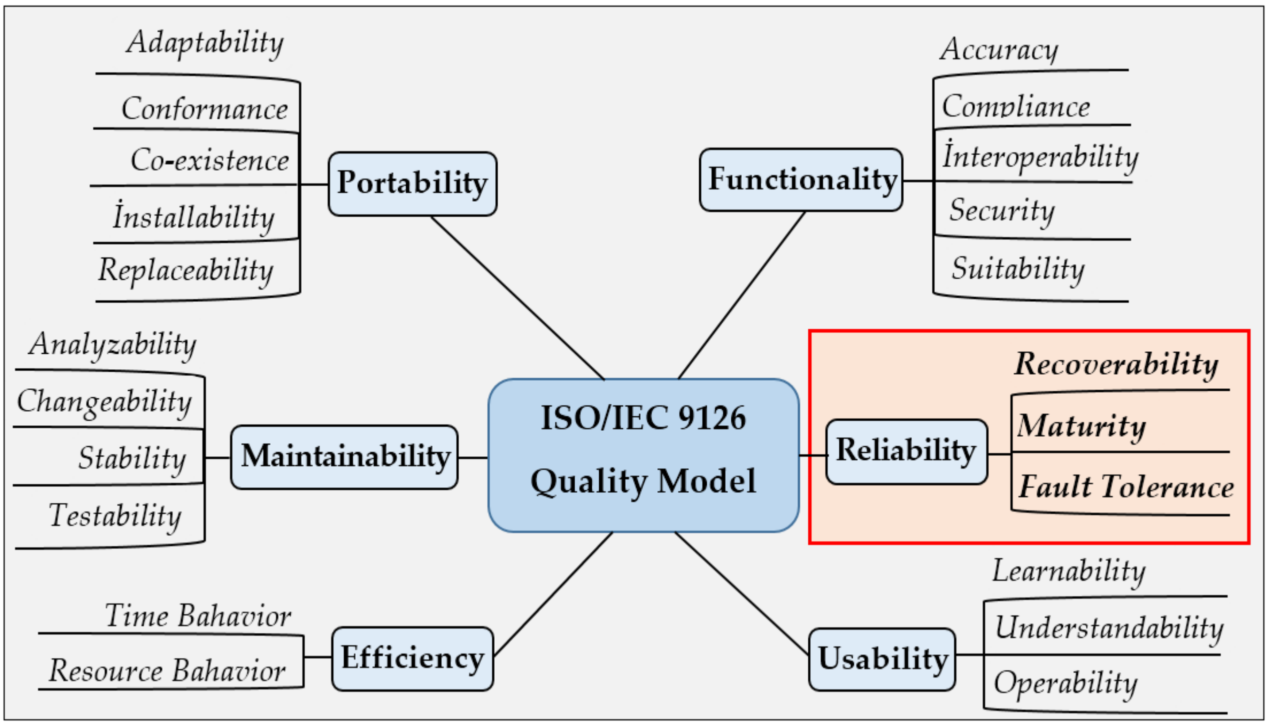

Fig. 1. ISO 9126 quality model

- comfortable and understandable user interface;
- possibility of data sharing with other information systems;
- fast operation;
- possibility of the software system expansion

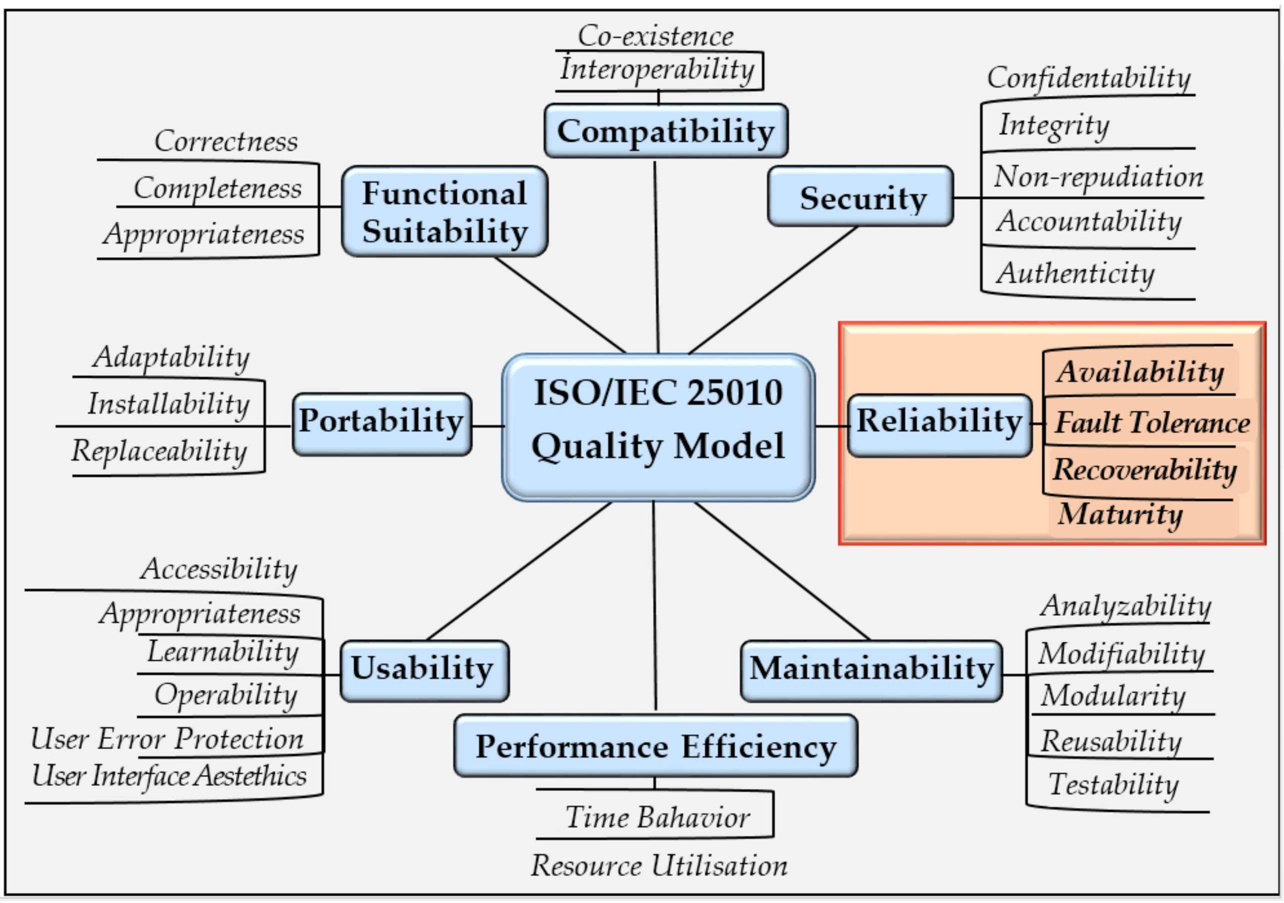

Fig. 2. ISO/IEC 25010 quality model

5. Reliability of software systems

There are two approaches to evaluating software reliability: qualitative and quantitative. In the quality approach, standards are based on a system of requirements defined by organizational normative documents. These requirements are checked when assessing software reliability. In the quantitative approach, software reliability is calculated based on various mathematical models. However, given the uneven distribution of errors and the fact that program failure is an unstable process, it is impossible to accurately calculate reliability based on these classic models (Shuman, Celinski-Moranda, Goel-Okumoto, Corkoren, etc.).

Among the quality characteristics of software systems, software reliability is considered a key parameter, as it quantifies the failures that can bring down even the most powerful system, ensuring that software systems run correctly and that there are no unexpected failures. Most users consider reliability to be a fundamental aspect of quality. Measuring the reliability of software systems is an important step in developing quality software (Rashid, Mahmood & Nisar, 2019). Software Reliability Engineering (SRE) is adopted as a best practice or standard by a number of companies (AT&T, IBM, NASA, Microsoft, etc.). High requirements are placed on the reliability of software developed for real-time systems, medical equipment, aviation, space industry, etc.

Different organizations provide various definitions of software system reliability:

- Software reliability is the probability of the software to run without failure for a specified period of time and in a specified environment (ANSI/IEEE, 1991).
- Software reliability is the ability of a system or component to perform required functions for a specified period of time

under specified conditions (Smidts, Stutzke, & Stoddard, 1998).

According to the ISO/IEC 25010 (2011) standard, software reliability is the degree to which a system, product, or component performs certain functions over a certain period of time, that is, the ability to maintain its performance when used under certain conditions. The sub-parameters of reliability according to this standard are listed below:

- *Maturity* - compliance of the system, product or component with the requirements for reliability in normal operation.
- *Availability* - degree to which the system, product or component is functional and available at the time of need.
- *Fault tolerance* – degree to which the system, product, and component maintain their operational status despite the presence of errors in the software.
- *Recoverability* - degree to which a system, product, or component can recover data or restore the desired operating mode of the system in the event of an accident.

Software failures are not caused by component failure or deterioration, they are caused by undetected errors in software. Unfortunately, even in enterprises where experienced and professional web developers and testing specialists work, it is impossible to eliminate 100% of errors in software. Because the functions software performs are increasing day by day, the volume and complexity of the software code is also growing accordingly.

Software system failures differ from hardware failures. The reliability of a software system does not change over time unless the programmer makes changes or a new version is developed. Hardware wears out over time, but a software system doesn't. The time dependence of software and hardware reliability is illustrated in Fig. 3 and Fig. 4, respectively.

As can be seen from the figures, the reliability of the hardware decreases over time and the number of failures increases after a certain moment. However,

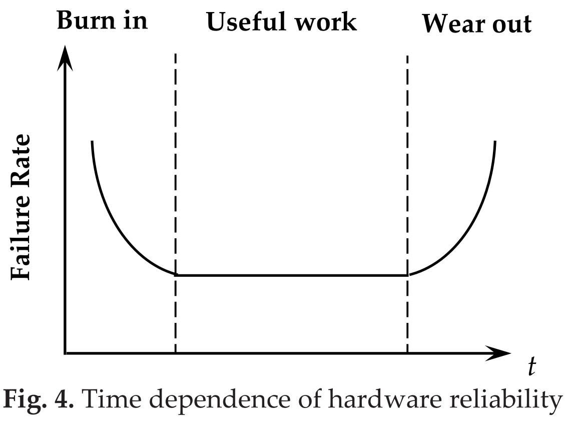

Fig. 4. Time dependence of hardware reliability

the number of failures in software does not increase over time, but rather decreases. When the software system is updated, the number of failures increases due to the introduction of new errors and decreases again as the errors are corrected. That is, the reliability of software systems directly depends on the number of errors in it (Van Driel et al, 2014).

Measuring and predicting reliability helps to increase the lifetime of any software. Table 1 presents similarities and differences between hardware and software reliability (Ghodrati et al., 2015).

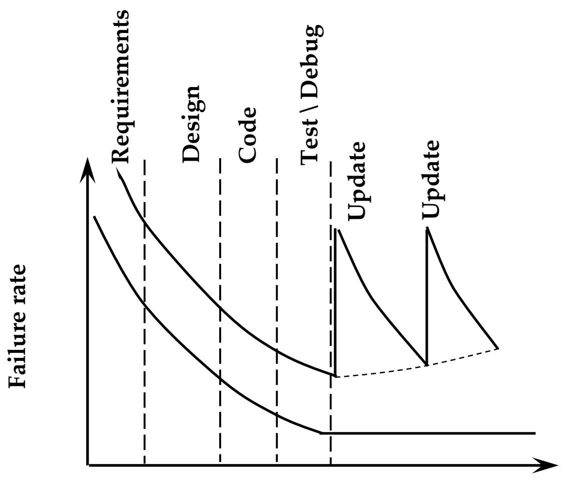

Fig. 3. Time dependence of software reliability

Table 1. Similarities and differences between software system and hardware reliability

Software reliability	Hardware reliability
Interfaces are conceptual (changeable)	Interfaces are visual
Design doesn't use standard components	Design uses standard components
It does not wear out over time. New failures may appear after the "old" code is updated.	Failure may occur due to wear and other stress/strain related errors.
Failure occurs during the execution of an incorrect logical path. Reliability increases by detecting and eliminating errors. Depending on the duration of work, the intensity of failures is decreasing or stable, unpredictable.	Depending on the duration of work, the intensity of failures can be decreasing, stable or increasing. Component failures occur based on physical laws.
Failures are the result of constructive errors.	Failures occur as a result of projecting, manufac-ture and service.

Maintenance is not provided	Equipment can become more reliable as a result of maintenance.
Environmental conditions do not affect the reliability of the software. Internal environmental conditions (memory, low tact frequency, etc.) affect reliability.	Reliability depends on environmental conditions.
Standard components are not used during development of software.	Standard components are used in the construction of the equipment.

Unlike hardware reliability, software reliability is difficult to measure because software is not a material product and its essence cannot be well understood. To achieve high quality, it is necessary to improve manufacturing, risk and configuration management processes.

Industry 4.0 offers new opportunities for engineers and programmers to build modern systems and apply innovations to smart devices and tools. While these innovations contribute to infrastructure development for the industry, they also create new and unknown failure mechanisms, unknown threats and risks. This directly affects the reliability of software systems (Gokhale, 2007). The increase in the complexity of the software code makes it difficult to develop error-free and full-quality software systems.

Reliability of software systems is a function of errors remaining in the software system after commissioning. In other words, the evaluation of the reliability of software systems depends on the number of uneliminated errors and errors remaining in the program. If new errors are not added when they are corrected, the reliability of the software system increases during operation. The more intensively the software is operated, the more intensively the errors remaining in it are detected, and the reliability and, accordingly, the quality of the system increases. An error-free software is completely reliable. However, since it is impossible to build 100% error-free software, it is practically impossible to achieve absolute reliability. Undetected errors appear over time under certain conditions during system maintenance and operation (Jatain & Mehta, 2014).

The length of the software code in currently created complex software systems is measured in millions of lines. The increase in software complexity has led to an increase in the number of errors in it, and the average number of errors per thousand lines of untested software code varies in the range of 10-50 (Nayyar, 2019). During the development of software systems, the presence of errors in the program code is inevitable, searching for and eliminating these errors requires a lot of effort and time. A software reliability error is an error or distortion accidentally introduced during the processing of the software code that causes failures or limitations of functionality during the operation of the software. Software failure is when the program stops working completely, and/or for a certain period of time causing data loss, and/or stops working and requires the program to be reloaded.

A large number of errors in software leads to the following consequences:

- Software reliability, as well as quality become low;
- Users are dissatisfied with the software performance.

The IEEE 610.12 (1990) standard gives an explanation of the concepts of error, defect, and failure in software:

- *Error*: negligence of software developers, misunderstanding of requirements, incorrect design of requirements, etc. resulting in problems in the software code. Errors in software code are made by software engineers, programmers, analysts and testers.
- *Defect or fault*: program errors that cause the software system to fail to perform its functions. Defects or faults are detected during software system testing, but not all defects detected during testing cause errors (for example, spelling errors in comments). Testing experts also use the term Bug.
- *Failure:* inability of the software system (component) to perform the expected functions within certain requirements or stop its work (Fig. 5.). Software failures can result in major disasters in critical systems.
- *Failure rate:* frequency of failures during the testing or operation of the software system.

Errors can occur at all stages of the software life cycle. Errors in design, development and operation can lead to different results. Errors that occur during the design and writing of the software code are eliminated to some extent during the testing process. Errors remaining during operation are gradually detected and corrected. However, there are errors that seem insignificant or unnoticeable, turn into serious errors over time and the applicability of the software becomes complicated. This type of errors are serious problem bearers, because they manifest themselves after long-term operation of software systems. Software projects that seem successful at first glance can be stopped or a code is rewritten due to such errors. This leads to an increase in budget costs in several times. Leading software companies are working on these problems.

Error management to save resources, improve test efficiency, user satisfaction, and software reliability, is one of the pressing issues facing software engineering.

Special attention is paid to accompanying software in critical information systems, collecting, analyzing and classifying errors that occur in the work process.

Errors detected during testing and after implementation of the software system are classified according to the source of occurrence, degree of severity and priority of editing (table 2). The severity of errors is determined by the degree to which they affect the functionality or performance of the software system. The more these errors affect the operation of the system, the higher the severity level. The priority of software system errors is the sequence in which they are eliminated after they are detected. The severity of a bug or error is often determined by the software system quality engineer and the priority by the project manager. 3 groups of defects are distinguished according to their priority (table 3.) (Defect Severity And Priority In Testing With Examples And Difference, 2023).

Typically, the priority of error correction depends on its severity. Errors with critical priority should be fixed first. In some cases, errors of low severity need to be fixed urgently (for example, incorrect display of flag colors).

Table 2. Classification of software errors by severity

No	Type	Description
S1	Critical	Errors defined as critical do not allow the operation or testing of the software system to continue (for example, shuts down the program).
S2	Major	These types of errors cause a serious problem in the operation of the software system, the main functions do not work.
S3	Minor	These types of errors do not seriously affect the operation of the program, but cause serious inconvenience to the user (for example, errors in the interface).
S4	Low/ Trivial	These types of errors are not very noticeable or do not affect the functionality of the software system (e.g. grammatical and spelling errors).

Table 3. Classification of software errors by priority

No	Type	Description
P1	High	The error seriously affect the operation of the system, the software system cannot be used without its elimination, so it must be corrected immediately.
P2	Medium	The bug fix is not urgent, it may be fixed in the next version.
P3	Low	The effect of the error on the operation of the system is not so serious, it can be corrected after eliminating other errors, or even if it is not corrected, it does not cause a problem.

Determining the cause and type of errors found in the testing and work process and classifying them facilitates their management, reduces testing costs and prevents repetition of failures (Fig. 5).

The main causes of faults in the software system are as follows:

- Complexity of the problem realization;
- Short work schedule;
- Limited budget funds;
- Frequently changing requirements;
- Lack of professionalism and work experience of the software team;
- Complexity of technologies used in modern software industry;
- Errors in the technologies used in programming;
- Integration of modules and components into the system;
- Improper preparation of documentation;
- etc.

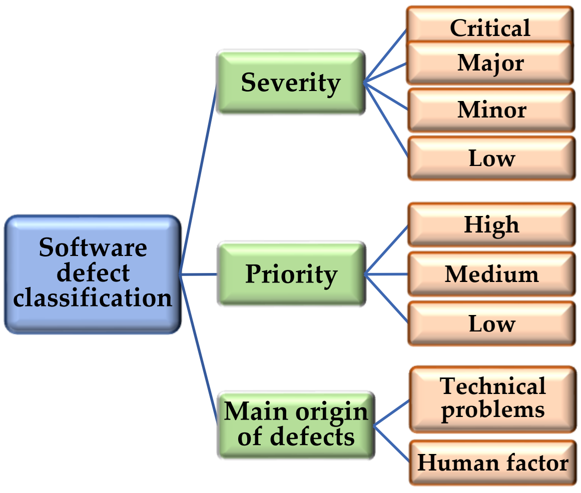

Fig. 5. Classification of errors in software systems

6. The main factors affecting the reliability of software systems

Most research on the reliability of software systems focus on the analysis of the internal quality of the system and did not take into account the environmental factors that directly affect the reliability of software systems during the development, testing and implementation stages. Development of software systems is a product of intelligence. For a comprehensive assessment of reliability, both internal and environmental factors must be taken into account. In recent years, various studies have been conducted on the effect of environmental factors on the reliability of software systems (table 4).

In 2000, Zhang and Pham (2000) surveyed 32 such factors, taking into account all stages, human factors, and hardware parameters covering the software life cycle. Empirical studies were conducted at 13 software companies. Managers, system engineers, programmers, testers participated in the survey. In the study, the impact of all factors on the reliability of software systems was analyzed and ranked. In the next study, they analyzed the relationship between them to reduce the number of factors (Zhang et al, 2001). 15 years later, Zhu et al (2015) repeated their first survey at 20 software companies. Table 5 presents the results of both surveys.

Table 4. Basic studies on the effect of environmental factors on the reliability of software

Authors	Research object	Country	Factor
Zhang and Pham (2000)	Based on 23 inquiries at 13 different companies	USA	32
Zhang et al. (2001)	Based on 35 inquiries at 13 different companies	USA	32
Zhu et al. (2015)	Based on 35 inquiries at 20 different companies	USA	32
Zhu and Pham (2017)	Based on 45 inquiries at 20 different companies	USA	32
Özcan et al. (2020)	Based on a survey of 70 experts	Turkey	32

Ozcan et al. conducted a survey of 70 experts in the Turkish software industry with the participation of managers, software engineers, testers, analysts and others to determine the factors affecting the reliability of software systems.

All other studies in this area were conducted in the United States. This was the first survey conducted in the second country. The survey results were analyzed and ranked using statistical methods. The result of this study does not differ from previous years much (table 6). Most of the first 10 factors listed in previous studies were also included in the top ten here (Ozcan et al, 2020).

From these studies, it appears that the main factors affecting the reliability of software systems are testing processes, software code complexity, documentation, modern programming technologies, and the human factor.

Table 5. The factors affecting software reliability

N	Factors	
	Zhang and Pham (2000)	*Zhu et al. (2015)*
1.	Program complexity	Frequency of software specification change
2.	Programmer skills	Testing effort
3.	Testing coverage	Testing environment
4.	Testing effort	Testing coverage
5.	Testing environment	Software complexity
6.	Frequency of specification change	Programmer skills
7.	Testing methodologies	Percentage of modules reused
8.	Requirements analysis	Relationship of detailed design and requirements
9.	Percentage of reused code	Testing methodologies
10.	Relationship of detailed design and requirements	Domain knowledge
11.	Level of programming technologies	Programmer organization
12.	Documentation	Program workload
13.	Program workload	Testing resource allocation
14.	Testing tools	Work standards
15.	Programmer organization	Requirements analysis
16.	Domain knowledge	Human nature
17.	Programming difficulty	Development management
18.	Design methodologies	Programming difficulty
19.	Human nature	Amount of programming effort
20.	Development management	Level of programming technologies
21.	Testing resource allocation	Testing tools
22.	Amount of programming effort	Documentation
23.	Program categories	Volume of program design documents

24.	Work standards	Design methodology
25.	System software	Development team size
26.	Volume of program design documents	Programming language
27.	Development team size	Program categories
28.	Programming language	Processor
29.	Processor	Telecommunication devices
30.	Telecommunication device	System software
31.	Input/output devices	Input/output devices
32.	Storage devices	Storage devices

Table 6. Top 10 factors affecting software reliability according to a survey conducted in Turkey

No	Factors
1.	Testing coverage
2.	Testing effort
3.	Testing environment
4.	Domain knowledge
5.	Program complexity
6.	Program workload (stress)
7.	Frequency of program specification change
8.	Average number of years in software development
9.	Level of programming technologies
10.	Human nature

7. Models improving software reliability

A measure of software reliability is the degree to which errors are detected and fixed. The software reliability modeling aims to:

- calculate the remaining time for the manufacture of the software product;
- evaluate the reliability of the product at the time of its release.

An accurate assessment of reliability provides both manufacturers and users with some degree of assurance that the software system will run successfully. Estimating the number of bugs remaining in a software system and predicting the time of the next failure are critical factors in deciding whether to release a software product or how long to test it. (Van Driel et al, 2021). Reliability modeling is the process of developing models to create error-free and reliable software systems. Numerous mathematical models have been developed for numerical evaluation of reliability of software systems and error management. These models are called software reliability growth models (SRGM). The number of failures and the time of their occurrence are recorded during the testing and operation of the program. Software reliability growth models are applied to predict the time of the next failure in the software system and the number of further errors to be detected based on this recorded historical data (Nagar & Thankachan, 2012).

In the field of detailed study of the reliability of software systems, the main issue is the study of reliability growth models and the development of new ones with the application of modern technologies. SRGM is a tool for evaluating the reliability of a software system and providing information about product quality to users (Lee et al, 2022). In classical reliability models, the reliability assessment of software systems is mainly based on the following parameters:

1. internal structural metrics of the program code (number of modules, subsystems, inter-module interfaces, operators, cycle complexity, etc.);
2. metrics detected during testing (time dependence of failures, number of failures, number of errors, time to find errors, etc.):

• ***Mean Time to Failure (MTTF)***

It is defined as the time unit between consecutive rejections. If the average failure-free operation time is not 150, then a software failure may occur every 150 time units.

• ***Mean Time to Repair (MTTR)***

Following a software failure, it takes some time to detect and fix the error. MTTR is the average time required to find and fix an error.

• ***Mean Time Between Failure (MTBF)***

$$MTBF = MTTF + MTTR$$

MTBF=300 means that when a failure occurs, the next failure can occur after 300 hours.

• ***Rate of occurrence of failure (ROCOF)***

It is the number of failures per time unit. ROCOF=0.02, i.e., 2 failures can occur every 100 time units.

• ***Probability of Failure on Demand (POFOD)***

POFOD is the probability that the system will fail when a request for service is sent. If POFOD=0.1, then one out of every 10 requests sent for the service may result in a failure.

• ***Availability (AVAIL)***

It is the probability that the system is available for use at the current time. If AVAIL=0.995, then the system will be available in 995 time units out of 1000. If the system has not worked for 4 hours out of 100 hours, its availability is 96%.

- ***Number of undetected errors;***
- ***Trial period (calendar time, processor time)***.

3. environmental factors affecting the reliability of software systems and the structure of input data.

The probability of the software system failure is measured in the interval [0,1]. This indicator of a system with high reliability is close to 1, and on the contrary, the closer this value is to 0, the lower the reliability. Reliability can also be measured by failure-free time. For example:

- The probability of 50 hours of uninterrupted operation of the information system is 0.99;
- The system controlling the trains movement breaks down every two years.

The first reliability model was proposed by Hudson in 1967. Shortly thereafter, Shuman applied the methods and terms used in hardware reliability to the calculation of software reliability in his 1968 book Probabilistic Reliability. In the same year, the term "software engineering" was first used at the NATO conference, and a separate session focused on the problem of software reliability. Software reliability engineering has its roots in structural, hardware, and electrical engineering. The Goel-Okumoto and Celinski-Moranda models are among the first published robust models in this field. As failures are detected and corrected in a software system, its reliability increases and the intensity of crashes decreases (Fig. 6.) (Cusick, 2019).

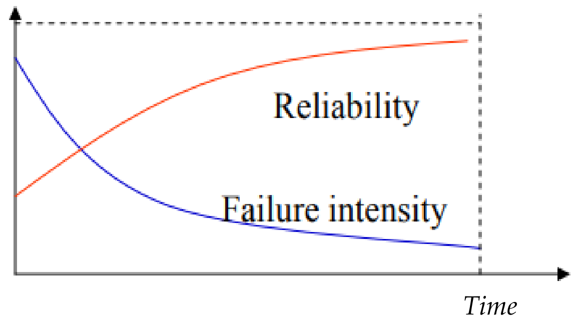

Fig. 6. Relationship of reliability to failure intensity

The basic principle of SRGM is to determine whether the reliability of the software system has increased sufficiently to meet the expected requirements by applying data about the time of occurrence of software failures or test results. Software engineers can evaluate the reliability of software systems using these models. Project managers determine the testing period and the release time of the software product.

Since 1972, hundreds of such models have been developed, but very few of these models have been tested on real data, and only a small number of models have been applied. Two groups of software systems reliability growth models are distinguished: parametric and non-parametric models. Parametric models are built on the basis of certain assumptions. 3 types of these models are available: non-homogeneous Poisson processes (NHPP), Markov models and Bayesian models. NHPP models are widely used in evaluating the reliability of software systems. According to their characteristics, software reliability models are grouped into 2 classes: deterministic and probabilistic. Deterministic models are based on the number of different operators and operands in the program code. Probabilistic models treat the occurrence of failures and the elimination of errors as probabilistic events. The first NHPP model, which inspired the development of others, was developed by Goel and Okumoto in 1979.

Although NHPP models are widely used, they are based on certain limitations or assumptions about the nature of software errors and failures. Therefore, choosing an appropriate model based on program characteristics is a complex issue. Accurate data is needed for both evaluation and forecasting, for which errors, failures, and the time of their occurrence detected must be accurately recorded during testing and operation. There is no model that accurately predicts all data, different models can predict well for a given dataset, and the best model is selected based on the comparison of the results of several models.

Over the past 50 years, hundreds of statistical and analytical models have been developed based on certain constraints and assumptions. However, as these models have different predictive capabilities for different data, there is no single model appropriate for all data. To overcome this problem, the development of non-parametric models based on intelligent technologies has been preferred in recent years.

8. Developing a conceptual model

Since the causes of faults in software systems are diverse, the demand for developing and evolving new solutions, algorithms, models and methods for the production of a reliable software system is increasing. To evaluate the reliability of software systems, it is not enough to develop methods using only the results obtained during testing. Table 5 identifies 32 factors affecting the reliability of software systems and arranges them according to their rating based on empirical experiments. In this study, the first 6 factors (documentation, testing, software complexity, programmer skills) are investigated and a comprehensive approach for reliability assessment is proposed. Fig. 7 provides a conceptual model of this approach. Minimizing errors in software code is one of the most pressing issues to ensure reliability, and the following comprehensive approach is required to implement this issue:

- Prevention of faults;
- Detection and elimination of faults;
- Ensuring fault tolerance;
- Development of fault prediction and reliability improvement models;

The first group includes principles and methods aimed at minimizing or completely eliminating faults. This group includes principles and methods whose purpose is to prevent errors in finished software code. They can be divided into the following categories:

- Depending on the purpose of the case, the team working on the software should be chosen correctly, because the main source of faults is the human factor;
- Minimizing the complexity of the program code, because the intensification in complexity leads to an increase in the number of undetected errors;
- Proper drafting of documentation.

Error prevention is the most optimal solution for developing reliable software. However, despite these measures, it is not possible to develop fault-free software. The source of faults in software is the professionals who work on it. In other words, they are specific people with individual characteristics, professionalism, talent and experience (Ferreira et al, 2023).

The second group includes the development of new approaches and methods for detecting errors during software testing. Errors are detected and eliminated during the testing of software systems. The increase in the complexity of software systems has increased the demand for the development of modern testing methods. Software testing is an effective technical tool that ensures the quality of software products by detecting errors. Understanding the cause and type of errors makes them easier to manage, reduces testing costs, and helps reduce the recurrence of defects by retraining software developers. Thus, the classification of software errors is a central process. Moreover, with the increase in the number of software systems and the increase in system complexity, the number of defects has increased to such an extent that it is impossible to accurately and quickly classify them with traditional methods. The development of machine learning has facilitated the automation of the testing process (Bayramova, 2020).

Many experts involved in the testing process have emphasized the importance of integrating artificial intelligence into software testing processes. Enterprises organizing testing processes with the application of artificial intelligence models and

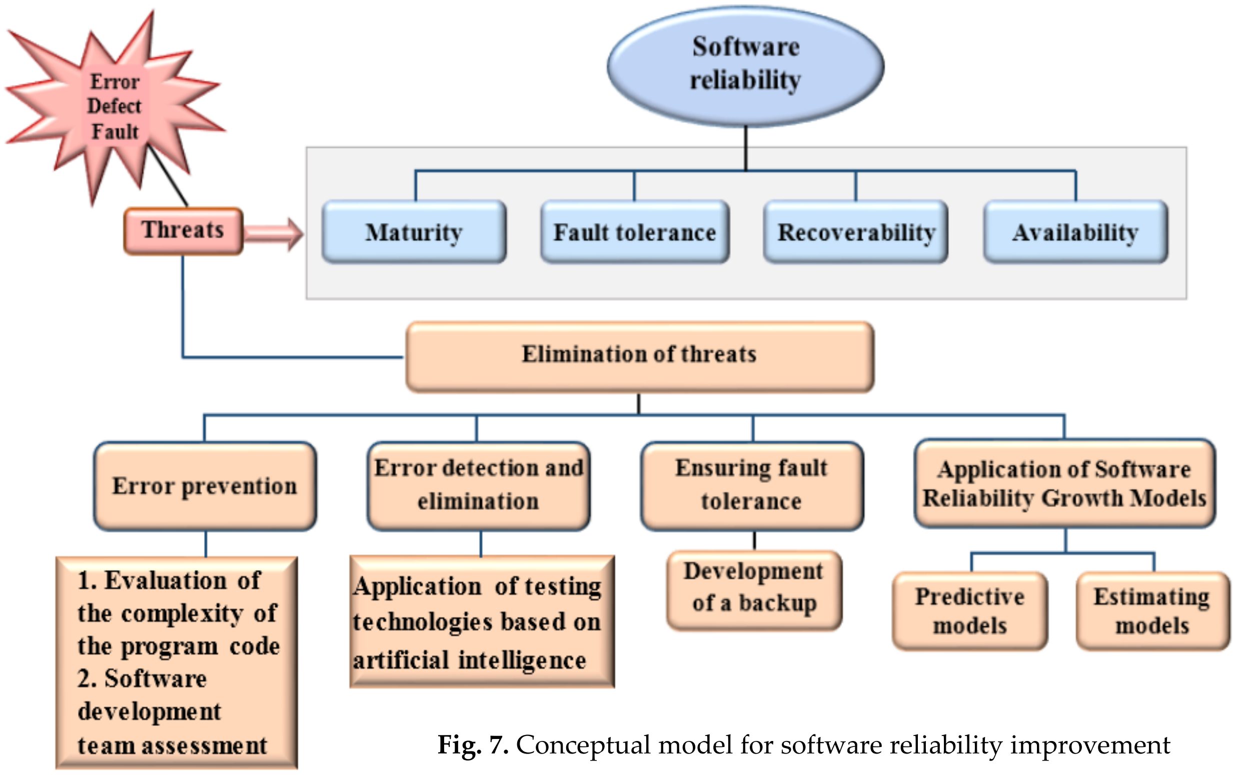
Error
Defect
Fault
Threats
Software reliability
Maturity
Fault tolerance
Recoverability
Availability
Elimination of threats
Error prevention
Error detection and elimination
Ensuring fault tolerance
Application of Software Reliability Growth Models
1. Evaluation of the complexity of the program code
2. Software development team assessment
Application of testing technologies based on artificial intelligence
Development of a backup
Predictive models
Estimating models

Fig. 7. Conceptual model for software reliability improvement

methods have already entered a new era, their competitiveness has also increased. New methods and models based on artificial intelligence have been developed for efficient processing of big data.

The application of artificial intelligence technologies in the testing of software systems provides more accurate results than traditional testing methods and decreases the time spent on testing, reducing the software development time (Kazimov et al., 2021).

The third group includes the development of new methods for automatic recovery of software when certain errors occur as a result of remained software errors. Resilience is the ability of software systems to continue operating under any circumstances. The purpose of software system stability methods is to prevent inactive errors from turning into active errors. Backup versions of software systems are created to ensure their resiliency. The purpose of the methods from this group is to maintain the operation of the software system despite the presence of errors. When an error occurs in the main program, control is transferred to the backup program. The disadvantage of these methods is that the cost of the software system increases, it is difficult to detect errors in the testing process. This method is used in particularly critical systems.

The fourth group includes the development of methods and algorithms for predicting errors in software modules and assessing reliability. Software reliability modeling is one of the most active areas of software engineering (Nagar & Thankachan, 2012).

9. Conclusion

Nowadays, medicine, transport, atomic energy, and many other critical areas have become dependent on software systems, which has further increased the demand for high-quality software products. In order to guarantee the quality of software systems, it is necessary to accurately assess reliability, which is one of its main indicators. A lot of realizations have already been done in this area, and various methods and models have been developed to evaluate the reliability of software systems. The significant increase in errors in modern software systems is a fundamental problem of software engineering, and the prevention, prediction and timely detection of these errors is one of the most urgent issues.

To evaluate the reliability of software systems, this study proposed a comprehensive approach to this problem by solving issues such as the evaluation of complexity of the software code and the potential of each team member working on the software project, the prediction of faulty software modules based on the metrics of the internal structure of the software code, prediction of the number of undetected errors and the duration of their occurrence based on the errors detected during testing and operation.

Overall, 3 groups of problems can be emphasized in the assessment of reliability of software systems:

- Lack of a unique methodology for developing software systems;
- Lack of a unique methodology for testing software systems;
- Lack of a unique approach to problem area analysis.

To ensure the software reliability, the following issues must be addressed:

- Determination of the factors affecting the software reliability;
- Application of software reliability enhancement methods in the design and operation processes;
- Improvement of available methods;
- Development of new methods based on artificial intelligence technologies for software evaluation and forecasting.

References

Abdallah, M., Jaber, T., Alabwaini, N., & Abd Alnabi, A. (2019). A proposed quality model for the Internet of Things systems. In 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, 23-27.

Alaswad, F., & Poovammal, E. (2022). Software quality prediction using machine learning. Materials Today: Proceedings, 62, 4714-4720. https://doi.org/10.1016/j.matpr.2022.03.165.

Albeanu, G., Madsen, H., Popențiu-Vlădicescu, F. (2020). Computational Intelligence Approaches for Software Quality Improvement. Reliability and Statistical Computing: Modeling, Methods and Applications, 305-317.

Alguliyev, R. M. & Mahmudov R. Sh. (2019). Sensitive personal data in the national mentality context and its security provision. . Problems of Information Society, №2, 117–128.

Al-Qutaish, R. E. (2010). Quality models in software engineering literature: an analytical and comparative study. Journal of American Science, 6(3), 166-175.

Bayramova, T.A. (2020). Analysis of software engineering standards. Problems of Information Society, 11(1), 83–95.

Bayramova, T.A. & Abbasova N.P. (2016). Verification and validation of software / «Questions of application of mathematics and new information technologies» III republican scientific conference, Sumgayit, December 15, 197-198.

Bayramova, T.A. (2022). Classification of software defects. Proceedings of the III international scientific conference on information systems and technologies: achievements and perspectives, Sumgayit, 256-258.

Bayramova, T.A. (2022). Analysis of Modern Methods for Detecting Vulnerabilities in Software for Industrial Information Systems // Cybersecurity for Critical Infrastructure Protection via Reflection of Industrial Control Systems, NATO Science for Peace and Security Series D: Information and Communication Security. - Amsterdam, 160-162.

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., Merritt, M. (1978). Characteristics of Software Quality. North Holland Publishing, Amsterdam, The Netherlands, 45-68, 169.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. Journal of systems and software, 80(4), 571-583. https://doi.org/10.1016/j.jss.2006.07.009.

Cristescu, M. P., Stoica, E. A., Ciovică, L. V. (2015). The comparison of software reliability assessment models. Procedia Economics and Finance, 27, 669-675. https://doi.org/10.1016/S2212-5671(15)01047-3.

Cusick, J. J. (2019). The first 50 years of software reliability engineering: A history of SRE with first person accounts. arXiv preprint arXiv:1902.06140.

Daniels, D., Myers, R., & Hilton, A. (2003). White box software development. In Current Issues in Safety-Critical Systems: Proceedings of the Eleventh Safety-critical Systems Symposium, Bristol, UK, 4–6 February 2003, 119-136. London: Springer London. https://doi.org/10.1007/978-1-4471-0653-1_7.

Febrero, F., Calero, C., & Moraga, M. Á. (2016). Software reliability modeling based on ISO/IEC SQuaRE. Information and Software Technology, 70, 18-29., https://doi.org/10.1016/j.infsof.2015.09.006.

Ferreira, F. H. C., Nakagawa, E. Y., dos Santos, R. P. (2023). Towards an understanding of reliability of software-intensive systems-of-systems. Information and Software Technology, 158, 107186.

Ghodrati, B., Hadi Hoseinie, S., Garmabaki, A. H. S. (2015). Reliability considerations in automated mining systems. International journal of mining, reclamation and environment, 29(5), 404-418.

Gokhale, S. S. (2007). Architecture-based software reliability analysis: Overview and limitations. IEEE Transactions on dependable and secure computing, 4(1), 32-40.

Gordieiev, O., Kharchenko, V., Fominykh, N., Sklyar, V. (2014). Evolution of software quality models in context of the standard ISO 25010. In Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland, pp. 223-232. Springer International Publishing. https://doi.org/10.1007/978-3-319-07013-1_21

IEEE 610.12-1990 Standard Glossary of Software Engineering Terminology. https://ieeexplore.ieee.org/document/159342

IEEE 730-2014 Standard for Software Quality Assurance Processes. https://ieeexplore.ieee.org/document/6835311

IEEE 1061-1992 Standard for a Software Quality Metrics Methodology. https://ieeexplore.ieee.org/document/237006.

In use qualities from ISO/IEC 25010. 3-4. https://www.irit.fr/recherches/ICS/projects/twintide/upload/435.pdf

ISO 9001 and related standards. https://www.iso.org/iso-9001-quality-management.html

ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model. https://www.standards.ru/document/3617603.aspx

Jalilian, S., & Mahmudova, S. J. (2022). Automatic generation of test cases for error detection using the extended Imperialist Competitive Algorithm. Problems of Information Society, 46-54.

Jatain, A., & Mehta, Y. (2014). Metrics and models for software reliability: A systematic review. In 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 210-214. IEEE. doi: 10.1109/ICICICT.2014.6781281.

Jharko, E. (2021). Ensuring the software quality for critical infrastructure objects. IFAC-PapersOnLine, 54(13), 499-504. https://doi.org/10.1016/j.ifacol.2021.10.498.

Kazimov, T. H., Bayramova, T. A., & Malikova, N. J. (2021). Research of intelligent methods of software testing. System Research & Information Technologies, 4, 42-52.

Kumar, A., & Gupta, D. (2017). Paradigm shift from conventional software quality models to web based quality models. International Journal of Hybrid Intelligent Systems, 14(3), 167-179.

Lee, D. H., Chang, I. H., & Pham, H. (2022). Software reliability growth model with dependent failures and uncertain operating environments. Applied Sciences, 12(23), 12383.

Maevsky, D., Kharchenko, V., Kolisnyk, M., & Maevskaya, E. (2017, September). Software reliability models and assessment techniques review: Classification issues. In 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2, 894-899. doi: 10.1109/IDAACS.2017.8095216.

McCall, J., Paul, K., Richards and F. Walters (1977). Factors in software quality: concept and definitions of software quality / Final Technical Report General Electric Company, 1, 25-31, 168.

McConnell, S. (2004). Code complete. Pearson Education. 952.

Mengmeng Z., Xuemei Z., Hoang P. (2015). A comparison analysis of environmental factors affecting software reliability, Journal of Systems and Software, 109, 150-160, https://doi.org/10.1016/j.jss.2015.04.083.

Miguel, J. P., Mauricio, D., Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. arXiv preprint arXiv:1412.2977.

Musa, J. D., & Everett, W. W. (1990). Software-reliability engineering: Technology for the 1990s. IEEE Software, 7(6), 36-43.

Nagar, P., & Thankachan, B. (2012). Application of Goel-Okumoto model in software reliability measurement. Int. J. Comp. Appl. Special Issue ICNICT, 5, 1-3.

Nayyar, A. (2019). Instant approach to software testing: Principles, applications, techniques, and practices. BPB Publications, India, 2019, 99-101, 368.

Ndukwe, I. G., Licorish, S. A., Tahir, A., MacDonell, S. G. (2023). How have views on software quality differed over time? Research and practice viewpoints. Journal of Systems and Software, 195, 111524.

Ozcan, A., Çatal, Ç., Togay, C., Tekinerdogan, B., Donmez, E. (2020). Assessment of environmental factors affecting software reliability: A survey study. Turkish Journal of Electrical Engineering and Computer Sciences, 28(4), 1841-1858.

Rashid, J., Mahmood, T., Nisar, M. W. (2019). A study on software metrics and its impact on software quality. Technical Journal, University of Engineering and Technology (UET) Taxila, Pakistan 24(1), 1-14.

Sahu, K., & Srivastava, R.K. (2019). Revisiting software reliability. Data Management, Analytics and Innovation: Proceedings of ICDMAI 2018, 1, 221-235.

Sahu, K., & Srivastava, R. K. (2020). Needs and importance of reliability prediction: An industrial perspective. Information Sciences Letters, 9(1), 33-37. https://digitalcommons.aaru.edu.jo/isl/vol9/iss1/5

Smidts, C., Stutzke, M., Stoddard, R. W. (1998). Software reliability modeling: an approach to early reliability prediction. IEEE Transactions on Reliability, 47(3), 268-278. doi: 10.1109/24.740500.

Standard Glossary of Software Engineering Terminology, STD-729-1991, ANSI/IEEE, 1991

The cost of poor software quality in the us: a 2022 report. (2023). https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/

Van Driel, W. D., Bikker, J. W., Tijink, M. (2021). Prediction of software reliability. Microelectronics Reliability, 119, 114074.

Van Driel, W. D., Schuld, M., Wijgers, R., Van Kooten, W. E. J. (2014). Software reliability and its interaction with hardware reliability. In 2014 15th International Conference on Thermal, Mechanical and Mulit-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE) Ghent, Belgium, 1-8. IEEE. doi: 10.1109/EuroSimE.2014.6813774

Yadav, H. B., & Yadav, D. K. (2017). Early software reliability analysis using reliability relevant software metrics. International Journal of System Assurance Engineering and Management, 8, 2097-2108. https://doi.org/10.1007/s13198-014-0325-3

Yang, L., Zhang, H., Shen, H., Huang, X., Zhou, X., Rong, G., Shao, D. (2021). Quality assessment in systematic literature reviews: A software engineering perspective. Information and Software Technology, 130, 106397. https://doi.org/10.1016/j.infsof.2020.106397

Zhang, X., & Pham, H. (2000). An analysis of factors affecting software reliability. Journal of Systems and Software, 50(1), 43-56. https://doi.org/10.1016/S0164-1212(99)00075-8

Zhang, X., Shin, M. Y., Pham, H. (2001). Exploratory analysis of environmental factors for enhancing the software reliability assessment. Journal of Systems and Software, 57(1), 73-78.