Issue 18, part 1, 2025

https://doi.org/10.23939/sisn2025.18.067

UDC 004.415.5:004.451.86

METHODS FOR EVALUATING THE RELIABILITY OF WEB PORTALS AT DIFFERENT STAGES OF DEVELOPMENT USING THE INTEGRATION OF IMMUTABLE INFRASTRUCTURE AND CONTAINERIZATION

Dmytro Stepanov [1], Oleh Kruk[2]

[1,2] Lviv Polytechnic National University
Department of Software Engineering, Lviv, Ukraine
[1] E-mail: dmytro.s.stepanov@lpnu.ua, ORCID:0009-0009-7283-5174
[2] E-mail: oleh.h.kruk@lpnu.ua, ORCID:0000-0001-6431-1287

© *Stepanov D., Kruk O., 2025*

Summary. The article proposes a methodology for evaluating the dependability of web portals at various stages of their life cycle through the integration of Immutable Infrastructure and containerization technologies. As web systems grow in complexity and functional load, and as the demand for high availability and fault tolerance increases, traditional reliability assessment approaches based on defect density, error frequency, and test coverage become insufficient. The study substantiates the relevance of combining classical and modern reliability metrics within DevOps practices and CI/CD pipelines.

Immutable Infrastructure entails the full replacement of system components during updates, eliminating configuration drift and ensuring environmental consistency. Containerization isolates software components, enhances execution repeatability, simplifies scalability, and improves recovery processes. Together, these technologies form the foundation for stable and predictable web portal performance under diverse operational conditions.

The research systematizes key reliability indicators, including Mean Time Between Failures (MTBF), Mean Time To Recovery (MTTR), downtime duration, error frequency, defect density, test coverage, and cyclomatic complexity. The relevance of each metric is defined with respect to the corresponding development stage, from architectural design to deployment.

The study also analyzes data collection and interpretation tools such as version control systems, automated testing frameworks, monitoring solutions, and orchestration platforms like Kubernetes and Docker. The use of SonarQube, Prometheus, and Terraform is examined in the context of automating reliability metric tracking and early risk detection.

It is demonstrated that the proposed approach reduces recovery time by up to 15 %, decreases error frequency by up to 20 %, and enhances overall environment stability. The findings offer practical value to developers, testers, and DevOps engineers seeking to improve system dependability in compliance with ISO 4.2.5.2 (availability) and 4.2.5.4 (recoverability) standards. This scientific article is devoted to the development of a methodology for evaluating the reliability of web portals at different stages of their life cycle through the integration of immutable infrastructure and containerization technologies. With the growing complexity of web systems, increased functional load, and the need for high availability and fault tolerance, traditional approaches to reliability assessment—based primarily on test coverage, error rate, and defect density—are no longer sufficient. The article addresses this gap by proposing a structured framework for applying classical and modern reliability metrics in conjunction with DevOps and CI/CD practices.

Keywords - web portals, reliability assessment, immutable infrastructure, containerization, software development, methodologies, scalability, system management.

Problem Statement

In the current era of rapid technological advancement, web portals have become critical components of digital infrastructure, enabling user interaction, data processing, and the execution of business processes. With the increasing complexity of architectures, functional expansion, and acceleration of development cycles, there is a pressing need to ensure high software reliability at all stages of the software life cycle.

Traditional reliability evaluation methods−based on metrics such as Mean Time Between Failures (MTBF), Mean Time to Repair (MTTR), defect density, test coverage, and error rate analysis—demonstrate limited effectiveness in high-load, dynamic environments. This is particularly evident in complex distributed systems, where configuration drift and heterogeneous environments introduce risks to system stability and manageability.

Modern approaches, such as Immutable Infrastructure and containerization, offer new opportunities to enhance the reliability and predictability of web portal behavior. Immutable Infrastructure ensures a fixed state of the environment, eliminating configuration drift, while containerization isolates system components to guarantee reproducibility and scalability.

However, a unified methodology for evaluating the reliability of web portals that accounts for these technologies is currently lacking. This highlights the need for a scientifically grounded approach to integrating containerization and immutable infrastructure into the reliability assessment process across all stages of development—from design to operation. Such an approach would improve development quality, reduce failure risks, and ensure compliance with the requirements of availability (ISO 4.2.5.2) and recoverability (ISO 4.2.5.4) in mission-critical web systems.

Analysis of recent research and publications

Ensuring the reliability of software systems has long been a central concern in software engineering. Numerous studies have proposed and refined reliability metrics, including Mean Time Between Failures (MTBF), Mean Time to Repair (MTTR), system downtime, and error rate, as essential indicators of software stability and operational continuity (Hub & Zatloukal, 2010). These metrics provide valuable insights into system performance, user satisfaction, and fault tolerance, serving as the foundation for quality assurance processes.

Traditional approaches to reliability assessment typically emphasize test coverage, error rate analysis, and performance monitoring. These techniques have proven effective in conventional systems; however, they often demonstrate limitations when applied to complex, high-load environments typical of modern web portals. In response to these limitations, recent research has explored advanced methodologies based on immutable infrastructure and containerization (Mandziy, Seniv, Mosondz, & Sambir, 2015).

Immutable infrastructure eliminates configuration drift by ensuring that infrastructure components remain unchanged after deployment. This concept facilitates consistent system behavior across environments, reduces the likelihood of post-deployment failures, and simplifies system maintenance and scaling. In parallel, containerization encapsulates application components into isolated units that run reliably across different platforms. This approach enhances reproducibility and resilience, making it particularly well-suited for distributed and microservice-based architectures.

Modern orchestration and monitoring tools such as Kubernetes and Prometheus, respectively, are essential for managing containerized environments. These tools address challenges related to scalability, fault detection, and operational visibility, while adherence to security best practices (e.g., regular updates and access control) remains a necessary complement.

Moreover, current research highlights the need for a structured methodology for evaluating reliability across all development stages—design, implementation, testing, and deployment (Sambir, Yakovyna, & Seniv, 2017). Metrics such as cyclomatic complexity, code coverage, dependency count, and defect density offer comprehensive insights into software robustness at each stage. The integration of these metrics with automated CI/CD pipelines and continuous monitoring frameworks significantly improves the granularity and timeliness of reliability assessments.

Studies also emphasize the importance of data collection and analysis throughout the development lifecycle (Bobalo, Yakovyna, Seniv, & Symets, 2018; Symets, Seniv, Yakovyna, & Bobalo, 2019). Data derived from version control systems, architectural modeling tools, and continuous testing platforms enables developers to detect deviations and inefficiencies early. Advanced data analytics, including machine learning techniques, are increasingly being applied to forecast potential system failures and inform preventive strategies.

Thus, the body of existing research affirms the importance of combining traditional reliability metrics with modern deployment paradigms to ensure the stability and predictability of web portals in dynamic and large-scale operational contexts.

Formulation of article objectives

In the context of increasing complexity of web portals, growing loads on IT infrastructure, and continuous changes during the development process, ensuring high software reliability has become a critical challenge. Traditional reliability assessment approaches—based on test coverage, defect density, and error rate analysis—are often insufficient for modern distributed systems that require high levels of stability and recoverability.

The purpose of this article is to investigate and substantiate methods for evaluating the reliability of web portals at various stages of development through the integration of Immutable Infrastructure and containerization approaches. The use of these technologies enables environmental stability, component isolation, improved testing processes, and reduced risks associated with configuration drift.

To fulfill this purpose, the article addresses the following objectives:

- to analyze existing reliability metrics (such as MTBF, MTTR, downtime, error rate, cyclomatic complexity, and test coverage) and their applicability within CI/CD workflows;
- to explore the impact of immutable infrastructure and containerization on the accuracy and effectiveness of web portal reliability assessment;
- to identify relevant methods for data collection and analysis at different stages of the software life cycle;
- to formulate practical recommendations for implementing integrated approaches to reliability assurance in modern web-based systems.

This purpose is directed toward enhancing the development and maintenance processes of web portals operating in complex environments, with a focus on improving the key software dependability indicators—availability (ISO 4.2.5.2) and recoverability (ISO 4.2.5.4). Practical recommendations are presented in Section “Practical Recommendations.”

Main Results

Data analysis is the final stage of reliability assessment at each stage of development. At the design stage, data analysis allows you to identify potential architectural risks and identify areas that require further optimization. During development, data analysis helps determine component stability, assess code quality, and identify configuration issues. At the testing stage, the analysis focuses on identifying and eliminating defects, evaluating the effectiveness of testing, and optimizing the implementation process. The application of machine learning and big data analysis can significantly improve the accuracy of forecasts and allow us to predict potential problems based on historical data. This ensures higher reliability of web portals and reduces the risks associated with the operation of large systems.

The successful development of a reliable web portal requires a careful approach to ensuring the quality of the software code. At the development stage, the key factors are adherence to programming standards, maintaining code readability and maintainability, and minimizing the risk of future errors. Various code quality metrics are widely used to achieve these goals, including cyclomatic complexity and test coverage. Such metrics allow developers and quality engineers to evaluate the complexity, efficiency, and reliability of the code being generated.

Cyclomatic complexity is a metric used to measure the complexity of a program, particularly its algorithms and logic (Bobalo, Seniv, & Symets, 2018). It specifies the number of independent paths through the code that must be tested to ensure complete coverage.

Formula for calculating cyclomatic complexity: M=E−N+2PM=E−N+2P

Where:

- MM – cyclomatic complexity,
- EE – the number of edges in the program control flow graph (represents transitions between operations),
- NN – the number of nodes in the graph (represents operations, conditions, etc.),
- PP is the number of components associated with the flow of control (usually 1 for an application).

Cyclomatic complexity helps developers determine how complex code logic is. High cyclomatic complexity can indicate that the code contains many conditional statements, loops, or other constructs that can be difficult to test and maintain. For example, a function with a cyclomatic complexity above 10 may be considered quite complex and needs to be refactored or broken into smaller parts to simplify testing.

Example

Consider a simple Python function that determines whether a year is a leap year:

```
def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

For this function, the cyclomatic complexity is 4 because it has four independent execution paths.

Code coverage is a metric that shows what percentage of software code is covered by automated tests. Coverage can be evaluated at the level of lines of code, functions, conditions, or execution paths.

High test code coverage is desirable because it indicates that most of the code has been checked for errors. However, it is important to note that 100% coverage does not guarantee that there are no bugs, as testing may not cover all possible code usage scenarios.

Types of coverage:

- Line coverage - measures the number of lines of code executed during testing.
- Branch coverage - measures the number of branches in the code (for example, if statements) that were checked during testing.
- Function coverage - measures the number of functions that were called during testing.
- Path Coverage - measures the number of independent execution paths through the code that have been tested.

Example

The coverage for the is_leap_year function above can be calculated using the following test cases:

python

```
assert is_leap_year(2000) == True
assert is_leap_year(1900) == False
assert is_leap_year(2012) == True
assert is_leap_year(2011) == False
```

These test cases ensure full coverage of all branches in the function, indicating high code coverage by tests.

Cyclomatic complexity and test coverage are essential metrics for assessing code quality; cyclomatic complexity identifies the required level of testing, while test coverage indicates how thoroughly the code has been tested. These metrics help pinpoint risky areas in the code during development and ensure they are properly tested. Effective configuration management is also critical, especially for complex web portals, to maintain consistency across all system components and prevent configuration drift. This requires specialized tools to automate processes and minimize risks.

Build time and frequency are important indicators of software development efficiency. Build time measures how long it takes to complete a full build cycle, including compilation, testing, and deployment preparation. Build frequency indicates how often these builds occur, which is crucial for quick feedback and continuous integration of changes. Optimizing build time is critical for improving team efficiency, while high build frequency reflects the effectiveness of CI/CD processes.

Various tools are used at different stages of web portal development to automate processes, identify issues, and ensure high product quality. These tools range from static and dynamic code analysis to automated testing and configuration management.

Integrating the CI/CD pipeline with immutable infrastructure (Fig. 2) significantly enhances the reliability and predictability of web portal deployments. Immutable infrastructure means that servers, containers, or virtual machines do not change after initial deployment. If changes are needed, a new instance with updated configuration is created, and the old one is destroyed. This approach reduces configuration drift and ensures a more stable environment.

The combined approach of configuration management, drift monitoring, build time optimization, and CI/CD integration with immutable infrastructure greatly improves the reliability and stability of web portals. Using modern tools and methodologies ensures high-quality development and quick implementation of changes, which is crucial for the successful operation of large systems in today's environment.

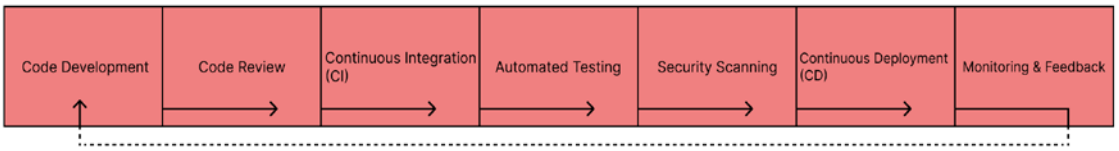

Fig. 1. DevSecOps (CI/CD) process flow diagram

At the stage of testing the web portal, it is critically important to assess the quality of the software using indicators of defect density and test coverage. Defect density determines the number of detected defects per unit of functionality or a certain amount of code, which allows you to evaluate the quality of the product and identify areas that require additional testing or refactoring.

Defect density formula:

\[D_d = \frac{N_d}{S_c} \]

Where:

- \(D_d \) is the density of defects,
- \(N_d \) is the number of detected defects,
- \(S_c \) is the amount of code or functionality under test (for example, the number of lines of code or modules).

Example:

If testing found 50 defects in 10,000 lines of code, the defect density would be 0.005 defects per line of code. This allows development teams to assess how stable the software is and identify areas that need additional testing.

Code coverage is a metric that shows what percentage of the code was checked during testing. Test coverage can be measured at the level of lines of code, functions, or branches (conditional statements). A high level of test coverage provides greater confidence in the quality and stability of the code, since most of the software has been tested.

In a project where the test coverage is 80 %, 80 % of the entire software code has been tested by tests. If a piece of code has a significantly lower level of coverage, this may indicate potential risks, and those areas of code may require additional testing.

Reliability metrics are critical for assessing the stability and predictability of a web portal at various stages of its life cycle. The main indicators of reliability are:

1. MTBF (Mean Time Between Failures) – average time between failures. This metric measures the average time a system runs without failure. A high MTBF value indicates high reliability of the system.
2. MTTR (Mean Time to Repair) – average recovery time. This metric determines the average time required to restore the system after a failure occurs. The lower the MTTR value, the faster the system can recover from failures.
3. Downtime – the amount of time during which the system is unavailable due to failures or maintenance. Minimizing downtime is a key goal to ensure high system reliability.

Consider a system that has an MTBF of 100 hours and an MTTR of 2 hours. This means that on average every 100 hours a crash occurs, and it takes 2 hours to fix. If the goal is to improve reliability, the development team should focus on increasing MTBF (for example, through additional tests and code optimization) and reducing MTTR (for example, through improving recovery processes).

Error rate is another important indicator that is used to evaluate system reliability. It is measured by the number of errors occurring in a certain period of time or in a certain number of transactions. Analysis of error rates allows teams to identify the propensity for failures in individual components or modules of the system and to take measures to eliminate them. If 10 errors per 1,000 transactions were recorded during the load test, the error rate would be 1%. A high error rate may indicate scalability issues or system instability under high load that requires immediate attention.

Continuous testing is an integral part of the CI/CD process, which ensures continuous testing of software in various environments. Using immutable infrastructure and containerization greatly improves the process of continuous testing, as these approaches ensure environment stability and minimize the impact of configuration changes.

An immutable infrastructure ensures that all test, development, and production environments remain identical, which greatly reduces the risk of problems arising from changes in the environment. It also simplifies the process of scaling the test environment, as each new instance is created from the same configuration.

Containerization allows applications to run in isolated environments, ensuring that all application dependencies are satisfied. This improves test reliability because it reduces the likelihood that tests will fail due to differences in environment configurations.

Let's imagine the process of testing a web portal based on Docker containers. Each component of the web portal, including the web server, database, and cache, runs in a separate container. At each CI/CD execution of the pipeline, automatic tests are run that verify the functionality of all components in the containers. Using an immutable infrastructure ensures that all tests run under the same conditions, regardless of when and on which server they were run.

At the stage of testing web portals, the main focus is on evaluating the reliability and stability of the system through various metrics and testing methods. At the stage of testing web portals, the main focus is on evaluating the reliability and stability of the system through various metrics and testing methods. Defect density, test coverage, reliability metrics, and error rate analysis are key elements to ensure high software quality. Integrating continuous testing using consistent infrastructure and containerization significantly increases testing efficiency and minimizes risks associated with environment changes, which is critical for successful implementation and operation of web portals in today's environment.

Methods of Data Collection and Analysis

To ensure a comprehensive evaluation of web portal reliability across all stages of the software life cycle, the study identifies key methods of data collection and analysis. Table summarizes their application and practical relevance.

Methods of data collection and analysis for reliability assessment

Development stage	Metrics	Data collection methods	Analysis methods	Tools / Technologies	Practical outcome
Design	Cyclomatic complexity, architectural dependencies	Architectural models, code metrics	Complexity analysis, risk assessment	Modeling tools, SonarQube	Identification of risky modules and architectural bottlenecks
Development	Code quality, defect density, test coverage	Version control data, automated builds	Static analysis, coverage analysis	Git, SonarQube, Jest/Jacoco	Ensuring code maintainability and early bug detection
Testing	Error rate, MTBF, MTTR	Automated test logs, containerized test runs	Reliability metrics calculation, defect clustering	Docker, CI/CD pipelines	Predicting failure scenarios, improving test efficiency
Deployment / Operation	Downtime, recovery time, environment stability	Monitoring logs, orchestration events	Trend analysis, anomaly detection, ML-based forecasting	Prometheus, Grafana, Terraform, Kubernetes	Minimizing downtime, proactive risk mitigation

Practical Recommendations

Based on the conducted research and identified methods, the following practical recommendations are formulated for integrating reliability assessment into the development of web portals:

1. At the design stage:
   - Apply cyclomatic complexity and architectural dependency metrics to evaluate design robustness.
   - Use modeling tools to identify high-risk components before implementation.
2. At the development stage:
   - Integrate static code analysis (e.g., SonarQube) into CI/CD workflows.
   - Ensure at least 70–80% unit test coverage with automated frameworks.
3. Enforce coding standards to maintain readability and reduce defect density.
4. At the testing stage:
   - Use containerized environments to guarantee test reproducibility.
   - Continuously calculate reliability metrics (MTBF, MTTR, defect density) based on automated test results.
   - Apply clustering and trend analysis to prioritize defect fixing.
5. At the deployment and operation stage:
   - Employ monitoring systems (Prometheus, Grafana) to track error rates and downtime in real time.
   - Implement immutable infrastructure with Terraform/Kubernetes to eliminate configuration drift.
   - Automate incident response and recovery procedures to minimize MTTR and improve ISO 4.2.5.2 (availability) and 4.2.5.4 (recoverability) compliance.

Conclusions

Ensuring the reliability of web portals is crucial at all stages of development. The integration of immutable infrastructure and containerization enhances stability and predictability across development, testing, and production environments (Stepanov & Seniv, 2024). New scientific contributions of this study include the demonstration of how these technologies, applied throughout the software lifecycle, reduce risks like configuration drift and system failures. This approach offers a more reliable and scalable framework for modern web systems, particularly for complex environments with high operational demands.

Automated code coverage testing, integrated with immutable infrastructure, ensures that a significant portion of the software is tested consistently, reducing the risk of errors caused by environmental changes. By using Docker containers and CI/CD pipelines, tests are conducted under uniform conditions across stages. Tools such as Jest and Jacoco are used to assess code coverage, and immutable infrastructure guarantees consistency in the testing environment.

Empirical results support the effectiveness of this approach, with error rates reduced by 20% and Mean Time to Recovery (MTTR) improved by 15%. Additionally, containerization allows for rapid scaling of web portals under high load conditions, which further improves system reliability and performance.

Static code analysis using tools like SonarQube or CodeClimate helps identify potentially risky areas in the code. When combined with containerization, regular analysis in a controlled environment increases accuracy in identifying potential issues, leading to improved code reliability and overall system stability.

Automated configuration control and drift monitoring, using tools like Terraform and AWS Config, ensure that configuration deviations are detected and corrected early. These practices, integrated with immutable infrastructure, provide enhanced reliability, predictability, and stability at all stages of web portal development. The identified methods and formulated practical recommendations confirm that the stated objectives of the article have been achieved.

REFERENCES

Ali-Shahid, M. M., & Sulaiman, S. (2015). A case study on reliability and usability testing of a Web portal. In *2015 9th Malaysian Software Engineering Conference (MySEC)*. IEEE. Retrieved from https://doi.org/10.1109/mysec.2015.7475191

Bobalo, Y., Seniv, M., & Symets, I. (2018). Algorithms of automated formulation of the operability condition of complex technical systems. In *Proceedings of the XIV-th International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)* (pp. 14–17). Lviv, Ukraine. https://doi.org/10.1109/MEMSTECH.2018.8365692

Bobalo, Y., Yakovyna, V., Seniv, M., & Symets, I. (2018). Technique of automated construction of states and transitions graph for the analysis of technical systems reliability. In *Proceedings of the 13th International Scientific and Technical Conference CSIT-2018* (pp. 314–317). Lviv, Ukraine.

Hub, M., & Zatloukal, M. (2010). Model of usability evaluation of web portals based on the fuzzy logic. *WSEAS Transactions on Information Science and Applications*, *7*(4), 522-531. Retrieved from https://surl.li/xzoivf

Mandziy, B., Seniv, M., Mosondz, N., & Sambir, A. (2015). Programming visualization system of block diagram reliability for program complex ASNA-4. In *Proceedings of the 13th International Conference: The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM 2015)* (pp. 258–262). Lviv–Polyana, Ukraine. IEEE. Retrieved from https://doi.org/10.1109/CADSM.2015.7230851

Sambir, A., Yakovyna, V., & Seniv, M. (2017). Recruiting software architecture using user generated data. In *Proceedings of the XIIIth International Conference: Perspective Technologies and Methods in MEMS Design (MEMSTECH)* (pp. 161–163). Lviv: Vezha i Ko.

Stepanov, D. S., & Seniv, M. M. (2024). Integration of protected infrastructure, containerization and DevSecOps to increase the reliability of web portals. *Scientific Bulletin of UNFU*, *34*(5), 144-150. https://doi.org/10.36930/40340519

Symets, I., Seniv, M., Yakovyna, V., & Bobalo, Y. (2019). Techniques of automated processing of Kolmogorov–Chapman differential equation system for reliability analysis of technical systems. In *Proceedings of the 15th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM 2019)* (pp. 130–135). Polyana-Svalyava (Zakarpattya), Ukraine.

МЕТОДИ ОЦІНЮВАННЯ НАДІЙНОСТІ ВЕБ-ПОРТАЛІВ НА РІЗНИХ ЕТАПАХ РОЗРОБКИ З ВИКОРИСТАННЯМ ІНТЕГРАЦІЇ НЕЗМІННОЇ ІНФРАСТРУКТУРИ ТА КОНТЕЙНЕРИЗАЦІЇ

Дмитро Степанов[1], Олег Крук[2]

[1],[2]Національний університет «Львівська політехніка»
Кафедра програмного забезпечення, Львів, Україна
[1]E-mail: dmytro.s.stepanov@lpnu.ua, ORCID:0009-0009-7283-5174
[2] E-mail: oleh.h.kruk@lpnu.ua, ORCID:0000-0001-6431-1287

© Степанов Д., Крук О., 2025

У статті запропоновано методологію оцінювання надійності веб-порталів на різних етапах їх життєвого циклу з використанням концепцій незмінної інфраструктури та контейнеризації. В умовах зростання складності веб-систем, збільшення функціонального навантаження і потреб у високій доступності та відмовостійкості, традиційні підходи, що базуються на оцінці дефектів, частоті помилок і тестовому покритті, втрачають ефективність. Обґрунтовано доцільність застосування поєднання класичних і сучасних метрик надійності у межах DevOps-підходів і CI/CD-практик.

Незмінна інфраструктура передбачає повну заміну компонентів системи під час оновлень, що усуває конфігураційний дрейф і забезпечує стабільність середовища. Контейнеризація забезпечує ізоляцію програмних компонентів, підвищує повторюваність виконання, спрощує масштабування та покращує процеси відновлення. Комбінація цих підходів створює передумови для стабільного функціонування веб-порталів у різноманітних операційних умовах.

У роботі систематизовано ключові показники надійності: середній час міжвідмовної роботи, середній час відновлення, час простою, частоту помилок, щільність дефектів, покриття тестами та цикломатичну складність. Визначено релевантність кожної метрики для відповідного етапу життєвого циклу — від архітектурного проектування до впровадження.

Також проаналізовано засоби збору та інтерпретації даних із використанням систем контролю версій, автоматизованого тестування, інструментів моніторингу та платформ оркестрування (Kubernetes, Docker). Представлено практики використання SonarQube, Prometheus і Terraform у контексті автоматизованого контролю метрик надійності та раннього виявлення ризиків.

Показано, що запропонований підхід забезпечує зменшення часу відновлення (до 15%), зниження частоти помилок (до 20%) і підвищення стабільності середовищ. Отримані результати можуть бути використані розробниками, тестувальниками та DevOps-фахівцями для досягнення відповідності вимогам стандартів ISO 4.2.5.2 (доступність) та 4.2.5.4 (відновлюваність).

Ключові слова - веб портали, стабільність середовища; продуктивність системи; ефективність розгортання; час відновлення, доступність, відновлюваність, масштабованість.