INFORMATION TECHNOLOGY, COMPUTER SCIENCE AND MANAGEMENT
ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ

UDC 004.05
https://doi.org/10.23947/2687-1653-2024-24-3-255-263

Original Theoretical Research

Quality Management in Software Development

Martin D. Birulia
EPAM Systems Sp z o.o., Kraków, Republic of Poland
✉ martinbirulia@gmail.com

EDN: JBGRGQ

Abstract
Introduction. The scientific literature examines various approaches to quality management in information technology (IT). The issues of identifying and correcting defects are worked out, and the possibilities for minimizing them are shown. There are materials on quality management in complex engineering processes. At the same time, there is no detailed description of quality management at each stage of the IT product life cycle, including testing. It should be noted that the coordination of software releases is closely related to quality management, but this process is rarely or fragmentarily considered in the literature. Additionally, the interprocess communication is not taken into account; therefore, there is no comprehensive understanding of quality management in the creation, testing and refinement of software. This study is designed to fill these gaps. The research is aimed at presenting a comprehensive approach that links the theory, practice and methods of software quality management.

Materials and Methods. Theoretical and applied literature on the subject were studied, analyzed, and reviewed. The author's professional experience in managing the quality of IT products was used. The practices of global suppliers of digital goods and services were taken into account. The author has used these materials and methods to study in detail the issues of software testing and code deployment.

Results. A comprehensive model of quality management in software development is elaborated, described and presented in the form of a diagram. Its interconnections with the project management model and the product life cycle, namely: analysis, design, development, testing, deployment, and support, are identified. Principles of quality management at each of these stages are specified. The processes and checks during code deployment are systematized and presented in the form of a diagram. Their features are shown in three environments: during development, testing, and production. The algorithm allows quality experts to build the sequence of actions to eliminate detected defects in the future, understand the situation when it is possible (or impossible) to deploy code and determine the moment when the software should be transferred to the user. In addition, the proposed scheme can be the basis for automating code deployment. The solution will reduce development time. As a result, the product will enter the market faster, which will speed up the payback of costs.

Discussion and Conclusion. The implementation of the model created within the framework of this scientific work into the production practice of IT companies presupposes strategic changes. Their implementation requires significant time and other resources; therefore, the overall transformation process should be divided into parts. The proposed approach is adaptable to the needs of different organizations and products. You can work with individual components to create an optimal plan for achieving quality management goals.

Keywords: code deployment, quality management in IT, IT product life cycle, IT product testing, software release

Acknowledgments. The author appreciates the leading software development organizations, such as EPAM Systems, Amazon, and Google, for their practices, which were analyzed under the preparation of this article. Special thanks to Project Management Institute for updating approaches to project management.

For Citation. Birulia MD. Quality Management in Software Development. *Advanced Engineering Research (Rostov-on-Don).* 2024;24(3):255–263. https://doi.org/10.23947/2687-1653-2024-24-3-255-263

Оригинальное теоретическое исследование

Управление качеством при разработке программного обеспечения

М.Д. Бируля

EPAM Systems, г. Краков, Республика Польша

✉ martinbirulia@gmail.com

Аннотация

Введение. В научной литературе рассматриваются разные подходы к менеджменту качества в сфере информационных технологий (ИТ). Проработаны вопросы выявления и исправления дефектов, показаны возможности их минимизации. Есть материалы об управлении качеством в сложных технологических процессах. Доказано, что работа с качеством цифровых продуктов требует в числе прочего прояснения вопросов качества кода. При этом нет детального описания управления качеством на каждом этапе жизненного цикла ИТ-продукта, включая тестирование. Отметим, что координация релизов (выпусков) программного обеспечения тесно связана с управлением качеством, однако данный процесс редко или фрагментарно рассматривается в литературе. К тому же не учитывается взаимодействие процессов, поэтому нет комплексного представления об управлении качеством при создании, тестировании и доработке программного обеспечения (ПО). Данное исследование призвано восполнить указанные пробелы. Его цель — представить комплексный подход, связывающий теорию, практику и методы управления качеством ПО.

Материалы и методы. Исследована, проанализирована и отреферирована профильная теоретическая и прикладная литература. Задействован профессиональный опыт автора в управлении качеством ИТ-продуктов. Учтены практики глобальных поставщиков цифровых товаров и услуг. Автор использовал эти материалы и методы для детальной проработки вопросов тестирования ПО и развертывания кода.

Результаты исследования. Сформирована, описана и представлена в виде схемы комплексная модель управления качеством при создании ПО. Выявлены ее взаимосвязи с моделью менеджмента проектов и жизненным циклом продукта, а именно: анализом, дизайном, разработкой, тестированием, развертыванием и поддержкой. Указаны принципы управления качеством на каждой из этих стадий. Систематизированы и представлены в виде схемы процессы и проверки при развертывании кода. Показаны их особенности в трех средах: при разработке, тестировании и производстве.

Обсуждение и заключение. Алгоритм позволяет специалистам по качеству выстроить последовательность действий для исключения в будущем выявленных дефектов, понимания ситуации, когда можно (или нельзя) развертывать код и определения момента, когда следует передать ПО пользователю. Кроме того, предложенная схема может быть базой для автоматизации развертывания кода. Решение позволит сократить время на разработку. Как следствие, продукт быстрее выйдет на рынок, что ускорит окупаемость затрат. Внедрение в производственную практику ИТ-компаний модели, созданной в рамках данной научной работы, предполагает стратегические изменения. Их реализация требует значительных затрат времени и других ресурсов, поэтому общий процесс трансформаций следует разбить на части. Предложенный подход адаптируется под нужды различных организаций и продуктов. Можно работать с отдельными компонентами, чтобы создать оптимальный план для достижения целей по управлению качеством.

Ключевые слова: развертывание кода, управление качеством в ИТ-сфере, жизненный цикл ИТ-продукта, тестирование ИТ-продукта, релиз программного обеспечения

Благодарности. Автор выражает благодарность ведущим организациям в сфере разработки программного обеспечения — EPAM Systems, Amazon и Google за их практики, которые анализировались при подготовке статьи. Особая благодарность компании Project Management Institute за обновление подходов к управлению проектами.

Для цитирования. Бируля М.Д. Управление качеством при разработке программного обеспечения. *Advanced Engineering Research (Rostov-on-Don).* 2024;24(3):255–263. https://doi.org/10.23947/2687-1653-2024-24-3-255-263

Introduction. Within the conditions of high competition on the software market, users focus not only on the marketing but also on the product utility, pay attention to the user-friendly interface, efficiency, and operational stability. All this should be taken into account by manufacturers and divisions of software companies that work with quality. There are enough theoretical and applied publications devoted to quality management and automation of these processes in open access. The authors consider innovations in this sphere, compare them with traditional practices.

Defect handling (specifically, their early detection and analysis) allows preventing errors at the software development stage. This approach reduces time, money and other resources spent on fixing logical, syntactic, compilation and other software errors. The quality of the product is improved, and its value in terms of reliability, ease of maintenance, and cost-effectiveness is increased [1]. In [2], quality management in complex processes is described. The author shows how to identify the relationship between technology and quality. According to his assumption, there is an information correspondence between probabilistic models of technology and quality. It is known from [3] that in the information technology (IT) sector, a 49% increase in budget leads to the creation of new quality management models. As an example, we can mention the ISO/IEC standards [4]. In [5], it is shown how quality management affects manufacturing processes.

An important and complex task is to automate the control of processes on which quality depends. The key form of control is statistical. It works with indicators that are critical to quality of the endproduct. These indicators are monitored and compared to target values. As a result, a conclusion about the statistical controllability or uncontrollability of the process is obtained [6].

Software development lifecycle analysis is covered in numerous sources, but there is no detailed description of managing each stage of the lifecycle, including testing [7].

A separate line of research is the relationship between the basic quality management tools and their impact on the operational, financial and market performance of the manufacturing company. At the same time, integrated use and management of such tools is an urgent task [8].

In [9], it is shown how different quality management practices affect innovation in ISO 9001 certified companies. Some studies focus on the incorporation of practice guidelines in product life cycle development. It is known from [10] that 60–80% of potential errors in innovative product development are related to misunderstood requirements. The authors of [11] argue that when discussing product quality, the issues of code quality, cost of achieving a given quality, and time to enter the retail market should be clarified. Continuous integration (or continuous delivery) of program code plays a special role in quality automation [12]. This is the process that provides continuous updating of the code during the development of a software product. In [13], recommendations are given on how to take into account the level of competition in the industry when releasing new versions of a product sequentially.

Each of these papers considers different elements of quality management. At the same time, the system and structural interprocess communication is overlooked. As a result, there is no comprehensive view of quality management in software development, testing and revision. This study is aimed at filling this gap. It describes an integrated approach linking theory, practice and methods of quality management in the IT project. It should also be noted that due to the flexibility of the proposed model it can be adapted to the needs of a particular manufacturer.

Role of Quality Management in Software Development. Quality is a complex category that can be considered from different standpoints: philosophical, social, engineering, economic, legal [1]. In this article, quality is discussed as a set of properties of a product or service that can, to one degree or another, satisfy the needs of the target audience [3]. The product is considered as the total result of different types of activities, and each has its own input data (parameters), which, upon the production process, turn into output (value). The product is a set of values. It is designed to meet needs that are determined in advance, during the marketing development of a product or service. The quality of the product is judged by how much the actual consumer satisfaction coincides with the planned one.

In the information technology, two groups of product requirements (user needs) are considered the basic ones: functional and non-functional. As on any market, in the retail sale of IT solutions, precisely structured work with value provides loyalty of the target audience, and therefore, increases the profitability of the release of software, applications and similar products.

Defective goods sold spoil the image of the manufacturer [8]. Negative feedback is spread online and in messengers. The loss of users leads to a deterioration in financial indicators. A quality crisis can result in the bankruptcy of the manufacturer.

In [6], it is shown that quality management through process management is associated with all five types of innovation. Let us list the factors that form them:

– new equipment and technologies;
– products with new properties;
– new raw materials;
– new production organization;
– new market channels.

Materials and Methods. The scientific research, whose results are summarized in the presented article, was based on the global practice of production and sale of software products and services. In addition, the literature devoted to the creation and promotion of IT solutions was studied. The theory has been correlated with the practice of the largest software developers. The theoretical part gives an idea of the specifics of different quality management methods. Examples of their practical use in product life cycle management are given. From this point of view, some approaches to project management are considered — flexible, cascade (second name — waterfall), and hybrid.

The organization and coordination of software releases is studied. It is closely related to quality management, but is rarely or fragmentarily considered in the literature.

The practice of implementing large projects is related to a well-studied theory of quality management. This allows us to determine precisely how large organizations use the theoretical basis to provide quality of the information products (and ultimately, their commercial success).

It should be noted that the author of this article is experienced in software products management, and his professional expertise was also used as research materials.

Types of software testing. As shown in [7], on the market (i.e., outside the manufacturing company), the perception of quality (as an element of competitive advantage) is determined in the process of product design, statistical control and feedback. The perception of quality within the company depends on what percentage of products pass all tests and ultimately do not require revision. This indicator is associated primarily with process management, and not with statistical control and feedback.

It is important that the elements of the system that affect quality are supported by senior management and aligned with human resource management. In addition, quality practices and indicators should be taken into account when setting up internal corporate relations, interaction with suppliers and other contractors.

Testing is performed to determine the quality of future and end products; therefore, it is important to understand what exactly is the object of tests. In this case, it is not enough to say: "product". This is too general a concept, it should be specified. Often, certain functional and non-functional requirements are imposed on the product. In the first case, we are talking about a set of functions that the system must perform in a certain way. Let us say, when typing an address, the corresponding page should appear in the browser bar. Selecting certain menu items on this page should lead to known results in advance. At the same time, there may be several website interaction scenarios — for example, for registered and unregistered users. Thus, most online resources, when registering, open up the opportunity to comment on posts, receive collections of materials, etc.

As the name implies, non-functional requirements are not related to the functions available to the user. However, they often provide the consumer with more significant benefits than functional ones. This could be a certain level of data protection, integrated analytics collection, compliance with legislation (e.g., on personal data protection).

Product compliance with functional and non-functional requirements is verified differently.

Figure 1 shows the functional requirements testing pyramid. Test types are ranked by importance, speed, cost, and success rate.

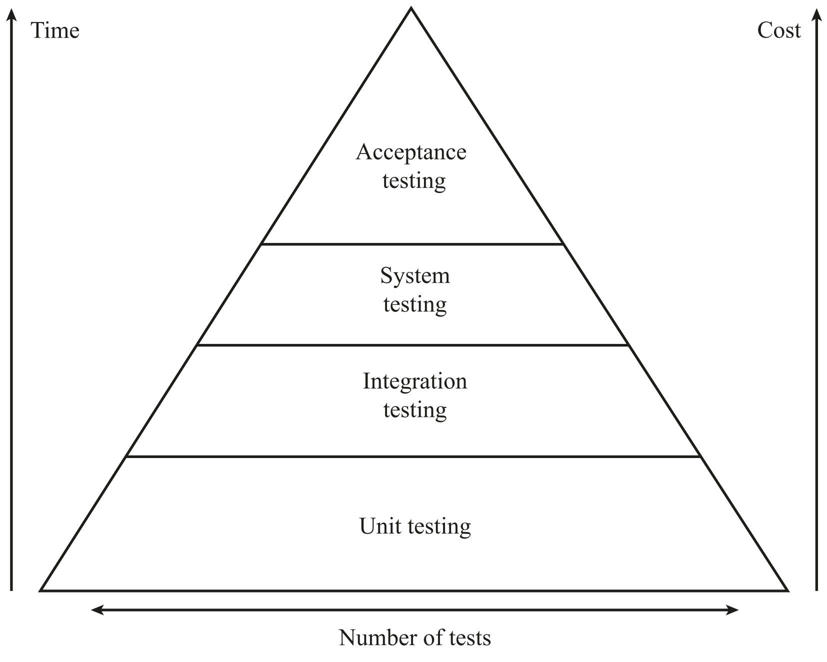

Fig. 1. Types of testing [14]

Unit testing, or modular testing, is a check of the correctness of individual modules of the program source code, individual functions of the application or service. This establishes whether the code is operational, whether it gives correct results with different inputs, and whether the logic matches the expected one. An example of such a test: when adding two numbers, their sum should be obtained — invariably. The input values required for testing are entered into the function. The unit test is passed if the expected result of the function is equal to the one actually obtained. The metric is calculated by comparing the number of lines of code that passed the test to the total number of lines of code in the repository. The best indicator is considered to be 80%. It shows that the logic of any function matches the expected one by 80% even before the next stage of testing starts. This means that problems are identified quite early and solved much cheaper and faster. Therefore, a unit test is the first step in the functional testing pyramid. Its absence (like any other type of testing) creates gaps, whose accumulation causes an unjustified increase in the cost of the project [15].

Integration testing. The next stage is testing the interaction of the components of the future application. The integration of modules is checked. These tests are more complex than unit testing, since the ability to perform sequential actions in various components is checked. Example: the user logs in — is redirected to the main page — places an ad. That is, several modules are tested at once. The basic options for integration testing are listed below.

"Big Bang". All components are assembled and tested together. This allows testing to be performed once after development. However, with a large number of modules, some may be overlooked, and this is the weak point of the method. In addition, the feedback loop increases, since ready-made solutions are tested when the development is complete. We should also note the difficulties with error localization. They are explained by the fact that it is necessary to analyze the entire development process (scenario) to find out the cause of the problem. This means that testing time will increase. The method is definitely convenient for small systems.

Incremental approach. Unlike the "big bang", this testing can start even if only two modules are ready for it, and add the rest as they are developed. In this case, it is easier to localize the initial errors, the feedback loop is shortened. There is no need to wait until all services are ready. This means that the testing will start and finish earlier. Removing defects will be cheaper. Incremental testing can be done in two ways: bottom-up and top-down.

Bottom-up integration. The least critical modules are tested first, then — the more important ones, and finally — the basic ones. Therefore, the upper components cannot be tested until there is data on the quality of the lower-level elements. This increases the time between the development and the start of testing.

Top-down integration. Testing starts with the most significant application layers. In this case, the testing of lower-level modules may not be completely adequate. However, even if this turns into problems later, they will be uncritical.

Hybrid integration. Modules of different levels are tested together. The success of such testing is determined by whether the developer understands the application architecture and can determine the optimal approach for a specific product [12].

System testing. All components of the program are tested as a single application. The specialist must ensure that the product correctly handles various scenarios and situations. This option may include testing non-functional requirements. This approach is based on requirements or on use cases. Whatever choices are made, the result will be test cases. Their successful completion confirms that the system works exactly as expected.

System testing requires considerable time, because as the product evolves, the number of possible use cases grows. Therefore, the software needs to be updated in a timely manner. This is done by the support service. With each product update, it is necessary to check whether the previously tested functions work. This process is called regression testing. If the tests worked before, but stopped working after the introduction of a new function, then there is a problem. The procedure of such checking sometimes takes up a huge amount of operation-use time for testing engineers, and they do not have time to write new tests. This explains the importance of automating tests that are performed manually. With this approach, testing is optimized to a simple analysis of the results of automatic checks. The best option is 100% automation. It is not always possible to achieve this figure, since some tests cannot be automated due to their complexity [16].

Acceptance testing. At the last stage, the product is tested by key specialists and customers. They check in real time to what extent their expectations and requirements have been met. It is determined if the functions work as intended, the interface is convenient, how likely errors and incorrect operations are, what problems the future user will face. As a result, the project is accepted or not accepted. Acceptance is the final stage of testing.

Fixing any bugs or defects found at this stage will be especially costly, since solving the problem will require going through the entire process from the very beginning, from development and unit testing to full system verification. Therefore, every effort should be made to avoid this situation at earlier stages. Non-functional testing checks the application for:

– performance;
– reliability;
– compatibility;
– security;
– utility;
– scalability.

More than 15 types of testing are used to check these properties. The most common ones are listed below.

Performance testing. The speed and efficiency of an application or system is assessed. The focus is on the speed of response to a user request. As an example, testing may include evaluating the loading time of the main page of a website. Performance targets are defined, and actual results are compared to them.

Load testing. This determines the maximum load a system can handle without failure or significant degradation in performance. Specifically, for a website, this means determining how many users can interact with the platform simultaneously before crashes or failures occur.

Fault tolerance testing. The ability of a system or application to remain operational during failures or unexpected situations is tested. The purpose of such testing is to detect vulnerabilities and develop measures to maintain continuous operation.

Compatibility testing. Compatibility of an application or system with other software, with different operating systems, browsers and devices is checked.

Security testing. The level of protection of a system or application from attacks and potential security threats is assessed. It is determined whether there are vulnerabilities. Protective mechanisms and the effectiveness of security measures are analyzed.

Disaster recovery testing. The ability of a system or application to recover from failures, crashes, or other emergencies is determined. The goal is to verify the effectiveness of recovery procedures and minimize downtime in the event of a disaster.

Research Results. Within the framework of the presented research, a comprehensive model of quality management in software development was formed. Its interrelations with the development life cycle, releases, costs, and the project management model (agile, waterfall or hybrid) were described.

Quality management process in the software development life cycle. Standard digital product development consists of six production processes, or stages. Each of them should contribute to quality assurance (Fig. 2).

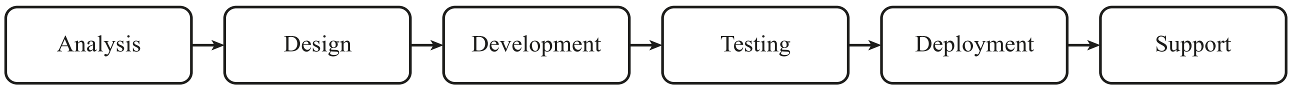

Fig. 2. Software Development Life Cycle [17]

Analysis and determination of product requirements. Problems are investigated, solutions are proposed taking into account the stated requirements. This process sets the key indicators for checking the endproduct. Depending on the development model, the strategy, approaches and testing tools are described and agreed upon. One of the documents is called “Product Vision and Capabilities”. It records the identified problems, notes the links with the business plan, describes the market situation, lists the main functional and non-functional requirements. The document is approved by the partner parties (i.e., representatives of the contractor and the customer). If necessary, adjustments are made to the text. At this stage, it is important to involve security and application architecture specialists who will help to accurately define non-functional requirements that are important for future testing.

Design of architectural and visual solution. Taking into account the requirements identified at the previous stage, an architectural solution is developed to provide their implementation, and the design of the user interface. All this is a guideline for the development team, who will test and, if required, adjust the product.

Development. Code is created to solve the tasks and problems of future users. Unit tests are written to check the logic of individual functions. The main goal is to achieve the required code coverage rate (at least 80%). However, other practices are also widely used, such as code reviews. In this case, the code is reviewed by other team members, and errors, shortcomings, omissions, and vulnerabilities are identified early in the development process. Reviewers can leave comments on specific lines of code, indicating the need for refactoring. In terms of product quality, the ideal scenario is dual code review, where at least two specialists unrelated to the author review and comment on the code.

In addition to unit testing and code review, it is necessary to use linters. These are automated tools that analyze code in real time, detect shortcomings, errors, and suggest ways to fix them. Other types of testing can be used depending on the strategy and requirements for the product.

Testing. Test engineers develop test scripts that are adequate to the tasks. As a rule, they are written manually, which slows down the implementation of this stage. It is advisable to automate the process, but this is not always possible. At this stage, specialists check and, if necessary, re-check the results of manual and automated tests. Particular attention is paid to cases where automated tests detect errors. The detected defects are ranked according to their importance to the system or users, and then are passed on to the developers for correction. After the defects are fixed, it is useful to perform a root-cause analysis (RCA) of the problem using the RCA method. This provides developing measures to prevent problems in the future, add new unit or integration tests. To fully understand the causes of the problem, it is recommended to use the “5 Why’s” technique. The method scheme looks like this: the problem is formulated and the question is asked: “Why did this happen?”. It is followed by an answer describing a certain fact or situation. The question is asked again: “Why did this happen?” And this is repeated five times. Five answers allow you to get closer to understanding the causes of the problem.

Testing is a large-scale process. The major stages are listed below.

1. Setting up and configuring the testing environment.
2. Writing and executing manual test scripts.
3. Automating previously written manual test scripts.
4. Checking the results of automated tests.
5. Maintaining up-to-date automated test scripts and fixing errors.
6. Recording defects.
7. Documenting defects.
8. Analyzing the causes of defects.
9. Developing steps to prevent future defects.

Deploying the application code. Software requires various environments for deployment. It is important to have a good understanding of how continuous integration and continuous deployment (CI/CD) maintain stable software quality with minimal effort. This is facilitated primarily by the automation of code deployment — the transfer of its changes from one environment to another. The process is divided into stages that depend on the product, organization, and accepted standards.

The degree of automation, the number and rigor of checks determine the expected quality of the product before the code is released to end users. Figure 3 shows a possible code deployment scheme. With a high level of automation, labor costs will be minimal, and only a few manual checks will have to be performed.

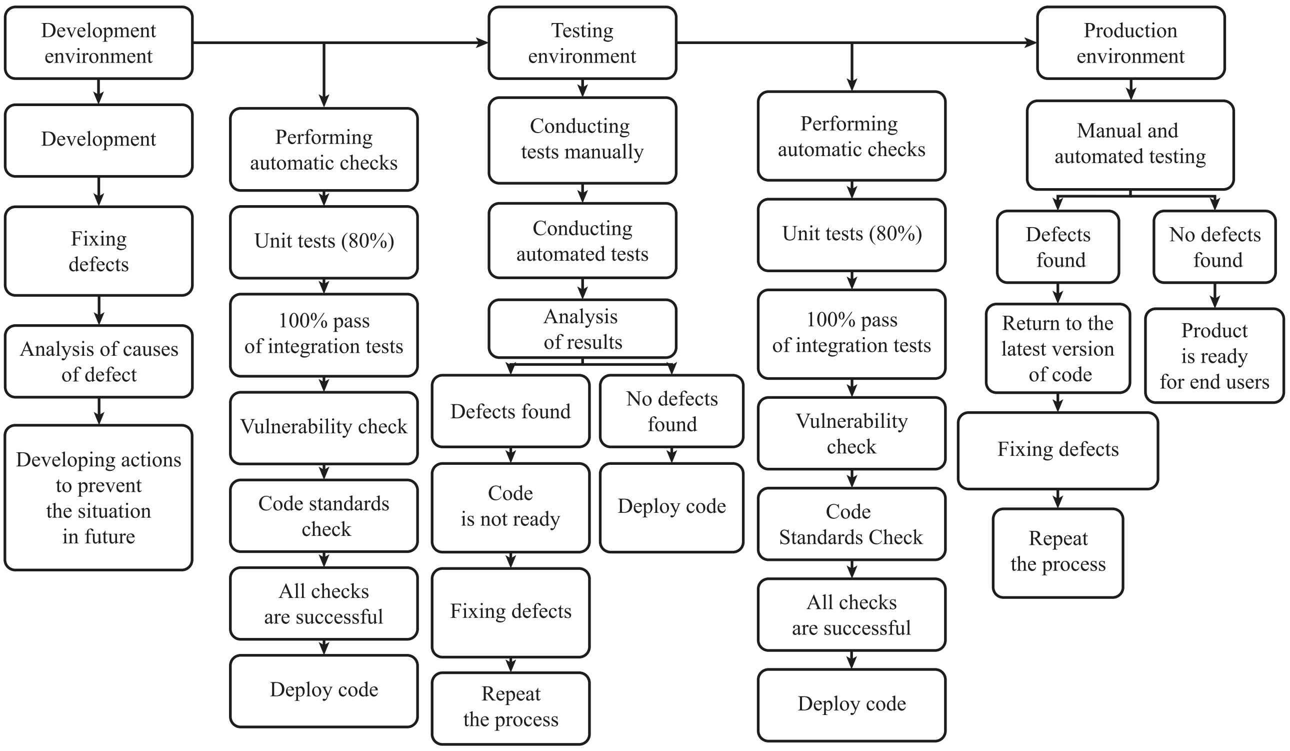
Development environment
Development
Fixing defects
Analysis of causes of defect
Developing actions to prevent the situation in future
Performing automatic checks
Unit tests (80%)
100% pass of integration tests
Vulnerability check
Code standards check
All checks are successful
Deploy code
Testing environment
Conducting tests manually
Conducting automated tests
Analysis of results
Defects found
No defects found
Code is not ready
Deploy code
Fixing defects
Repeat the process
Performing automatic checks
Unit tests (80%)
100% pass of integration tests
Vulnerability check
Code Standards Check
All checks are successful
Deploy code
Production environment
Manual and automated testing
Defects found
No defects found
Return to the latest version of code
Product is ready for end users
Fixing defects
Repeat the process

Fig. 3. Code Deployment Process and Checklist Diagram

Following the presented scheme, you can control the quality and automate the deployment of the code. This will reduce the development time, which means the product will enter the market faster.

Product support. At this stage, it is necessary to build feedback with users: conduct surveys, collect and analyze metrics. All this will be the basis for tasks and ideas for improving the software. Assumptions should be confirmed by A/B tests, i.e., comparison of product options. It is also important to provide feedback to the support team, which directly contacts users. These are the specialists who share valuable observations, convey wishes of the target audience, on the basis of which analysis is performed and new strategies are formed. Evidently, the process does not end even if the product is completely ready. New data appears for analysis, collection of requirements, and development of a digital product or service.

Discussion and Conclusion. The quality management model presented in this article can be adapted to the needs of various organizations and products. Its components can be further studied to create an optimal plan for achieving quality management goals. However, it is important to keep in mind the key factors for commercial software success: developers and testers must think critically, establish collaboration, and continuously improve processes.

The proposed model involves strategic changes, whose implementation requires a significant amount of time; therefore, the overall transformation process should be broken down into parts.

The presented material will be useful for management that administrates quality issues related to the company's overall strategy and optimization of individual processes.

It should be noted that achieving significant results requires a high degree of testing automation, a developed engineering culture, and significant costs for creating and maintaining infrastructure.

References

1. Suma V, Gopalakrishnan Nair TR. Defect Management Strategies in Software Development. In book: *Recent Advances in Technologies*. Vienna: Intec Web Publishers; 2009. P. 379–404. https://doi.org/10.48550/arXiv.1209.5573

2. Kuznetsov LA. Quality Management in Complex Processes. *Control Sciences.* 2007;(3):47–53.

3. Munoz M, Mejia J, Ibarra S. Tools and Practices to Software Quality Assurance: A Systematic Literature Review. In: *Proc. 13th Liberian Conference on Information Systems and Technologies* (*CISTI*). New York City: IEEE; 2018. P. 1–6. https://doi.org/10.23919/CISTI.2018.8399334

4. Carrozza G, Pietrantuono R, Russo S. A Software Quality Framework for Large-Scale Mission-Critical Systems Engineering. *Information and Software Technology.* 2018;102(3):100–116. https://doi.org/10.1016/j.infsof.2018.05.009

5. Vallabhaneni RS. Corporate Management, Governance, and Ethics Best Practices. Ch. 7. In book: *Quality-Management Best Practices*. Hoboken, NJ: Wiley; 2008. 456 p. https://doi.org/10.1002/9781119196662.ch7

6. Razmochayeva NV, Semenov VP, Bezrukov AA. Investigation of Statistical Process Control in Process Automation Tasks. In: *Proc. XII International Conference on Soft Computing and Measurements.* 2019;1:355–358. (In Russ.) URL: https://scm.etu.ru/assets/files/2019/scm2019/papers/7/355.pdf (accessed: 20.06.2024).

7. Nevlyudov ISh, Andrushevich AA, Evseyev VV. Software Development Life Cycle Analysis for Enterprise Information Systems. *Eastern European Journal of Enterprise Technologies.* 2010;6(8):25–27. (In Russ.)

8. Kaynak H. The Relationship between Total Quality Management Practices and Their Effects on Firm Performance. *Journal of Operations Management.* 2003;21(4):405–435. https://doi.org/10.1016/S0272-6963(03)00004-4

9. Dong-Young Kim, Vinod Kumar, Uma Kumar. Relationship between Quality Management Practices and Innovation. *Journal of Operations Management*. 2012;30(4):295–315. https://doi.org/10.1016/j.jom.2012.02.003

10. Ramasubbu N, Kemerer CF. Integrating Technical Debt Management and Software Quality Management Processes: A Normative Framework and Field Tests. *IEEE Transactions of Software Engineering*. 2019;45(3):285–300. https://doi.org/10.1109/TSE.2017.2774832

11. Alhassan A, Alzahrani W, AbdulAziz A. Total Quality Management for Software Development. *International Journal of Computer Applications.* 2017;158(5):38–44. URL: https://www.ijcaonline.org/archives/volume158/number5/alhassan-2017-ijca-912850.pdf (accessed: 20.06.2024).

12. Mohamed SI. Software Release Management Evolution — Comparative Analysis across Agile and DevOps Continuous Delivery. *International Journal of Advanced Engineering Research and Science*. 2016;3(6):52–59. URL: https://ijaers.com/detail/software-release-management-evolution-comparative-analysis-across-agile-and-devops-continuous-delivery/ (accessed: 20.06.2024).

13. Adelman D, Mancini A. Dynamic Release Management: A Market Intensity Approach. *Chicago Booth Research Paper*. 2016;(16–19):42. http://doi.org/10.2139/ssrn.2847264

14. Radziwill N, Freeman G. Reframing the Test Pyramid for Digitally Transformed Organizations. *Semantic Scholar*. URL: https://www.semanticscholar.org/reader/62a5c71b33437bc40e146a13a6fb95371b866262 (accessed: 22.06.2024).

15. Alves NSR, Mendes TS, Mendonca MG, Spinola RO, Shull F, Seaman C. Identification and Management of Technical Debt: A Systematic Mapping Study. *Information and Software Technology.* 2016;70(2):100–121. https://doi.org/10.1016/j.infsof.2015.10.008

16. Concas G, Marchesi M, Murgia A, Tonelli R, Turnu I. On the Distribution of Bugs in the Eclipse System. *IEEE Transactions on Software Engineering*. 2011;37(6):872–877. http://doi.org/10.1109/TSE.2011.54

17. Lemke G. The Software Development Life Cycle and Its Application. *Senior Honors Theses and Projects*. 589. Ypsilanti, MI: Eastern Michigan University; 2018. URL: https://commons.emich.edu/cgi/viewcontent.cgi?article=1588&context=honors (accessed: 22.06.2024).

About the Author:

Martin D. Birulia, project manager, EPAM Systems Sp z o.o. (114, Opolska Str., Krakow, 31–553, Poland)**,** SPIN-code, ORCID, martinbirulia@gmail.com

***Conflict of Interest Statement:* the author claimed no conflict of interest.**

The author has read and approved the final manuscript.

Об авторе:

Мартин Дмитриевич Бируля, менеджер проектов EPAM Systems Sp z o.o. (Польша, 31–553, г. Краков, ул. Опольска, 114), SPIN-код, ORCID, martinbirulia@gmail.com

Конфликт интересов: **автор заявляет об отсутствии конфликта интересов.**

Автор прочитал и одобрил окончательный вариант рукописи.

Received / Поступила в редакцию 30.06.2024
Reviewed / Поступила после рецензирования 20.07.2024
Accepted / Принята к публикации 27.07.2024