هيئة الاتصالات والفضاء والتقنية
Communications, Space & Technology Commission

Guidelines

for Software Quality Standards

cst.gov.sa

OCT 2024

هيئة الاتصالات والفضاء والتقنية
Communications, Space & Technology Commission

Table of Contents

1. Introduction

The Communications, Space, and Technology Commission (CST) is responsible for regulating and overseeing the technology sector in the Kingdom of Saudi Arabia (the Kingdom) in accordance with its amended regulation by Council of Ministers Decision No. (133) dated 21/5/1424AH (21/7/2003), which assigned it regulatory tasks in the information technology sector, as emphasized in item (seventh) of Council of Ministers Decision No. (292) dated 27/4/1441AH (24/12/2019), the CST exercises these powers with the aim of achieving the sector's regulatory objectives outlined in Article Two of the Communications and Information Technology Law (the Law), issued by Royal Decree No. (M/106) dated 2/11/1443AH (1/6/2022). These objectives include the development of information technology, improving markets, enhancing the maturity of services provided to end users, and establishing appropriate procedures for these goals.

Accordingly, the CST has developed the Guidelines for Software Quality Standards (the Guidelines) to enhance and improve software quality. Software quality has become a vital competitive element in the software industry, making it imperative for companies and entities striving for success in the digital economy to focus on high-quality and reliable software, which is a key factor for successful digital transformation. Additionally, high-quality software reinforces the Kingdom's reputation for providing leading digital solutions, contributing to making the Kingdom a leading regional and global digital hub.

This guideline aims to establish a set of practices that companies or entities involved in developing software or implementing software solutions can adopt to maintain high software quality. It serves as a reference guide, not a substitute for any controls, procedures, standards, rules, instructions, or regulations issued by the CST or relevant governmental bodies. It does not, in any way, serve as a reference for any legal procedures or responsibilities concerning individuals and parties involved.

2. Definitions

The following is a list of key definitions used in this document:

2.1 Compute Software:

A set of commands and instructions expressed in any language, symbol or sign, and which take any form. They can be used directly or indirectly in a computer to perform a function or achieve a result, whether these commands are in their original form or in any other form in which they appear through the computer, which includes the set of documents and exhibits attached and related to the computer program. This software can be cloud-based, accessed over a network, installed on computers or tablets, or embedded, such as in connected vehicles, medical devices, or more.

2.2 Software Documents:

The documents necessary for developing, analyzing, using, understanding, or supporting software include a set of attached documents related to the computer software. These include documents and materials from the design, analysis, and development stages, as well as testing and maintenance documents, process flowcharts, designs, schematics, algorithms, user guides, and any other documents that assist in understanding, operating, rewriting, or maintaining the software. It also includes documentation for all processes and procedures.

2.3 Software Quality:

The functional suitability of software, in comparison to software documents, includes performance efficiency, compatibility, ease of use, security and reliability, maintainability, and software integration. This is achieved through adherence to well-planned processes, effective risk management, ensuring stakeholder satisfaction, and implementing continuous improvement mechanisms throughout the software lifecycle.

2.4 Software Quality Assurance:

A set of ongoing activities to monitor all software engineering processes, methods, and work products, to ensure compliance with specified quality standards ensuring that all participants in the development process have implemented all procedures and processes correctly and effectively.

2.5 Beneficiary or User:

The person or persons, whether natural or legal, who have an interest or influence in the use of the software system, includes users and clients who utilize the software to achieve specific goals, as well as related parties such as partners and suppliers, and stakeholders such as shareholders, managers, and owners. Understanding the needs and expectations of these beneficiaries contributes to guiding the design and development process to ensure the success of the software project.

2.6 Software Developers:

The person or persons, whether natural or legal, who specialize in the design, development, or maintenance of software tailored to meet specific business needs, including entities that develop software internally.

2.7 Functional Requirements:

The ability of the software to meet the requirements and specifications outlined in the software documents in order to satisfy the needs or expectations of the beneficiary (for example, having an electronic page to register visitors).

2.8 Non-Functional Requirements:

The specifications that define the characteristics and constraints that the software system must possess, which are not related to specific functions or the direct behavior of the system. These requirements focus on quality and performance, covering additional needs that are not addressed by functional requirements (for example, the system's ability to process 1,000 requests per second).

2.9 Performance:

It refers to the efficiency and speed of the software's response to user requests.

2.10 Reliability:

It means the ability of the software to work stably and predictably without unexpected errors or malfunctions.

2.11 Interoperability and Compatibility:

It refers to the ability of the software to interact with other components and systems smoothly and without difficulties.

2.12 Security and Reliability:

It indicates that the software is well protected to ensure the integrity, confidentiality, and integrity of data and prevent security violations.

2.13 Maintenance:

Includes the ability to modify and update software easily and effectively without affecting the basic functions of the system.

2.14 Portability:

The ability to transfer software from one environment or system to another.

2.15 Usability:

The ease of using and learning the software.

2.16 Availability:

The availability of the software and the continuity of its operation to ensure access to the service.

2.17 Requirements Gathering:

¿The process of conducting research, exploring, and defining the functional and non-functional requirements of the system from the perspective of beneficiaries, clients, and other stakeholders.

2.18 Software Design:

The process of creating and defining the architectural structure, behavior, and components of a software system. It involves translating the requirements and needs of the beneficiaries into a guiding plan that directs the development process.

2.19 Software Development:

It refers to the actual process of creating a software system from design specifications. This includes writing code, integrating components, and conducting various testing and debugging activities to ensure that the software system operates correctly and meets the intended requirements.

2.20 Software Testing:

The process of evaluating and verifying that the software functions as intended based on the relevant software documents, along with checking for errors and gaps, and determining whether the software's output matches the desired expectations before it is installed and operated.

2.21 Deployment, Operation and Maintenance :

A set of activities and processes involved in launching and managing software systems, in addition to the activities and processes for maintaining and supporting the product after its release.

3. Scope of the Guidelines

This document can be beneficial for any natural or legal person engaged in the software industry in the Kingdom. The practice of the software industry includes usage, development, testing, and other core processes related to software.

4. Goals of the Guidelines

4.1 Assisting software developers and beneficiaries in implementing effective quality assurance and software testing that complies with local and international standards and controls.

4.2 Promoting the adoption of best practices in software quality to improve its functionality, security, and reliability.

4.3 Improving customer satisfaction by ensuring that the software meets the beneficiary's requirements and expectations.

4.4 Mitigating financial, legal, or reputational risks that may result or arise from software errors and vulnerabilities.

4.5 Promoting the growth and development of the software industry in the Kingdom to be capable of delivering high-quality software products to local and international markets

5. Quality Requirements During the Software Lifecycle

These guidelines, which can be followed throughout the software development lifecycle, have been developed in line with leading international standards and frameworks including ISO 25000, ITIL 4, CMMI, COBIT 2019, and others. The guidelines for each stage in the software life cycle are defined through the following stages, as shown in Figure 1.

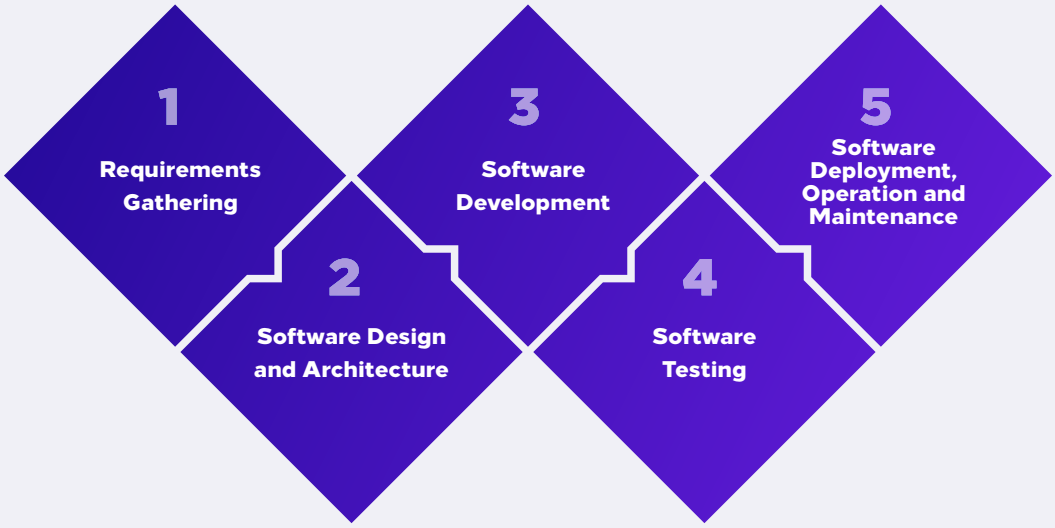

Figure 1: Software Lifecycle

Emphasizing the importance of adhering to the regulations related to security, reliability, data management and governance issued by the National Cybersecurity Authority and the National Data Management Office during the software development lifecycle.

5.1 Requirements Gathering phase

During the requirements gathering phase, software developers and beneficiaries can adopt the following practices, if applicable:

5.1.1 Identifying all stakeholders involved with the software product, which may include stakeholders, end beneficiaries, department managers, and any external systems with which the software will interact

5.1.2 Conducting interviews and surveys to extract detailed information about the expected purpose and specifications of the software product, using techniques such as meetings, surveys, or questionnaires that involve the identified stakeholders.

5.1.3 If the proposed software is designed to replace or interact with existing systems or applications; those systems and applications should be thoroughly analyzed to understand their operational strengths and weaknesses, as well as the required functionalities.

5.1.4 Documenting the gathered requirements in a clear, comprehensive, and continuous manner.

5.1.5 Creating prototypes to simulate and enhance the design, as modeling can help identify design errors and usability issues, providing stakeholders with a visual representation of the final product.

5.1.6 Prioritizing requirements by collaborating with stakeholders to determine the relative importance of each requirement, while considering factors such as business value or associated risks.

5.1.7 Reviewing and verifying the requirements carefully with all stakeholders to ensure accuracy and applicability before moving on to the next phase.

5.1.8 Requirements may change over time, so adopting an Agile development approach, using tools like Scrum or Kanban, enables iterative and flexible development. This approach allows for accommodating changes and ensures that the resulting software remains aligned with the beneficiaries' needs. The Agile approach applies to all phases of the development lifecycle, not just the requirements gathering phase.

5.2 Software Design and Architecture phase

During the software design and architecture phase, software developers and beneficiaries can adopt the following practices, if applicable:

5.2.1 The architectural design includes defining the overall structure of the software system, including high-level components and their interactions, as well as the distribution of functions.

5.2.2 Component design involves breaking the system into smaller, more manageable components and defining interfaces and data exchange between them.

5.2.3 User interface design includes creating the graphical user interface (UI) or user experience (UX) for the software system.

5.2.4 Database design entails structuring and organizing the database used by the software system.

5.2.5 Algorithm design involves developing algorithms and data structures to efficiently solve specific problems or achieve certain functionalities, which includes selecting appropriate algorithms and data structures according to the requirements and considerations like time complexity and space complexity.

5.2.6 Security and reliability design includes implementing mechanisms and measures to protect the software system from unauthorized access, data breaches, and other security and reliability threats.

5.2.7 Designing strategies and mechanisms for handling errors, exceptions, and unforeseen circumstances appropriately, which includes identifying exception handling mechanisms, error logging, and reporting.

5.2.8 Integration design encompasses designing interfaces and integration points with other systems or components, including defining data exchange formats, communication protocols, and application programming interfaces (APIs).

5.2.9 Engaging stakeholders, specifically business owners, clients, and experts, throughout the design phase to ensure feedback and expectations are addressed, minimizing the likelihood of major changes occurring later in the development cycle.

5.2.10 Conducting a comprehensive market survey to understand the target market, customer segments, trends, and competitive landscape with the aim of analyzing competitor products to identify gaps and competitive opportunities.

5.2.11 Defining the business model for the product and how to create added value and revenue, acquire customers, and maintain profitability, while considering other factors such as customer marketing costs, pricing strategy, distribution channels, and partnerships.

5.2.12 Identifying a revenue model that aligns with business objectives and the target market, considering options like subscription-based models, one-time purchases, free models, or ad-based models, while analyzing the pros and cons of each model and choosing the approach that best fits the product and target audience.

5.2.13 Conducting a feasibility study to assess the technical and economic viability of the proposed product, ensuring that the product design is user-friendly while considering factors like navigation ease, design, responsiveness, compatibility with different devices, and assistive technologies. Usability testing should also be conducted with real users to gather feedback on the design, identify areas for improvement, and verify that the software meets the beneficiaries' needs.

5.2.14 Determining the ownership model of the software product, which includes several options such as open source or proprietary (source code) or a combination of both, and evaluating the advantages and disadvantages, as well as the impact on revenue and market adoption associated with each licensing model.

5.2.15 Complying with relevant data protection regulations and international standards, and implementing high-level security and reliability measures to protect beneficiary data and address potential vulnerabilities.

5.2.16 Ensuring seamless integration with platforms or systems by identifying standard APIs or protocols that enable integration with external software components or third-party services, enhancing the value offered to the end user.

5.2.17 Designing the software product to ensure scalability to accommodate potential increases in the user base and data volume while improving performance through the implementation of efficient algorithms and appropriate data structures, considering system constraints such as response times and resource utilization.

5.2.18 Considering the "localization" requirements of the software product early in the design process when targeting global markets, taking into account language, cultural, and regional preferences, and supporting currencies, time zones, and varying local regulations where applicable.

5.3 Software Development phase

During the software development phase, software developers and beneficiaries can adopt the following practices, if applicable:

5.3.1 Code should be written in a way that is easy to understand without the need for extensive external comments. Good variable naming and breaking down large functions into smaller ones can aid in this. The simpler and clearer the code is, the easier it is to understand, debug, and maintain.

5.3.2 Adhering to best practices for the programming language used makes the code more efficient and maintainable, as each programming language has a guide for best practices.

5.3.3 An effective version control system, such as Git, should be used to manage source code and track changes. This facilitates collaboration among developers, eases code reviews, and helps prevent conflicts in code changes.

5.3.4 Frameworks should be kept up to date, and the latest versions should be used since frameworks regularly release updates that usually include performance enhancements, new features, and important security and reliability fixes.

5.3.5 Regular code reviews should be conducted to identify and address potential issues, ensure compliance with programming standards, and improve overall quality. Code review tools like Gerrit or Crucible can help facilitate this process.

5.3.6 Best practices for using open-source code should be followed, which include reviewing open-source licenses, conducting security and reliability assessments of open-source code, and attributing original authors and licenses in the software product.

5.3.7 Security and reliability practices should be applied throughout the development process. This includes code reviews for security vulnerabilities and reliability, using secure programming practices, and implementing security frameworks and standards in accordance with the security and reliability requirements from the National Cybersecurity Authority.

5.3.8 Error handling and exception management techniques and methods should be implemented by using appropriate logging mechanisms to capture errors and exceptions during software execution for debugging and troubleshooting.

5.3.9 The software development process should be documented, including the code base, functions, application programming interfaces (APIs), and configurations. Documentation helps in understanding the system, maintaining it, troubleshooting in the future, and collaborating with other team members.

5.3.10 Build files should be written to simplify and standardize the build process, saving time, reducing errors, and ensuring that the code can be consistently built, tested, and distributed across environments.

5.4 Software Testing phase

During the software testing phase, software developers and beneficiaries can adopt the following practices, if applicable:

5.4.1 Establish a dedicated quality assurance and testing team that is independent of the development team to avoid potential conflicts of interest and ensure neutral testing.

5.4.2 If outsourcing high-cost or high-risk projects, it is recommended to identify a third party that is independent of the development service provider to perform quality assurance and testing tasks.

5.4.3 Define a clear and documented testing methodology/guide that outlines the testing approach, objectives, and methodologies to be followed. This guide should cover various types of testing, such as functional testing, performance testing, security and reliability testing, and usability testing.

5.4.4 Create a defined and standardized testing procedure that includes specific activities such as test planning, test design, test execution, and test reporting to ensure consistency and repeatability in testing processes.

5.4.5 Leverage automation tools and frameworks to automate repetitive test cases that require significant time to complete, allowing tests to be executed faster and more reliably while reducing human errors and increasing test coverage.

5.4.6 Prepare an independent testing environment that matches the technical characteristics of the main environment (the hosting environment for the developed systems) to enable testers to conduct realistic testing and identify any issues or conflicts that may arise in the main environment.

5.4.7 Implement testing early in the development lifecycle, ideally conducting tests in parallel with development to help identify issues early on, thereby reducing the cost and effort required to fix them.

5.4.8 Adopt bug tracking systems to detect and track all issues during the testing process, helping to organize and prioritize bug fixes and ensuring that none are overlooked.

5.4.9 Incorporate data into the testing environment to simulate actual usage scenarios, which helps in identifying potential issues related to data integrity, performance, and scalability.

5.4.10 Involve end-users or potential customers in the testing process to provide valuable feedback, identify usability issues, and help verify whether the software meets their requirements and expectations. An independent testing environment for end-users or potential customers can be set up similar to the previously prepared testing environment for usability testing.

5.4.11 Conduct comprehensive regression testing to ensure that functionalities remain unaffected when updates or changes are made to the software. Regression testing helps in identifying any unintended side effects or bugs that may have occurred during development.

5.4.12 Prioritize test execution based on identified risks and the importance of system components to effectively allocate resources and focus on components that have a high impact on the system.

5.4.13 Utilize continuous integration practices to regularly integrate code changes and automatically run automated tests to identify integration issues early, ensuring regular testing throughout the development process.

5.4.14 Encourage early involvement of the testing team during the requirements gathering and analysis phase to identify potential issues early on and ensure testability of the software requirements.

5.4.15 Track and measure the scope of testing to ensure that all functionalities and use cases are thoroughly tested, through comprehensive coverage of code, requirements, and risk analysis.

5.4.16 Establish a procedure for effective test data management by identifying and providing suitable test data, ensuring data privacy, security, and reliability, and regularly updating test data to maintain test repeatability and effectiveness.

5.4.17 Perform performance testing to verify the software's ability to handle the expected load and pressure from users, helping to identify capacity limits and potential performance bottlenecks, as well as resource constraints.

5.4.18 Include security and reliability testing as a dedicated testing phase to identify vulnerabilities and risks related to data security, authentication, licensing, and encryption.

5.5 Software Deployment, Operation and Maintenance phase

During the software deployment, operation, and maintenance phase, software developers and beneficiaries can adopt the following practices, if applicable:

5.5.1 Implement automated deployment processes to simplify the release of software packages using tools such as Docker, Kubernetes, or Ansible to facilitate and streamline automation.

5.5.2 Utilize configuration management tools like Puppet, Chef, or Ansible to manage and automate configuration settings across different environments.

5.5.3 Apply a version and release management system to efficiently handle software product versions and releases, using tools like GitHub, GitLab, or Bitbucket for version control and release management.

5.5.4 Implement continuous integration from the development phase to automate the deployment of software packages to the production environment, ensuring a seamless and compliant deployment process.

5.5.5 Include disaster recovery plans and backup solutions for software and data to protect against potential failures or disasters, utilizing backup tools and distributed storage systems to ensure data integrity and availability..

5.5.6 Utilize monitoring tools to oversee software performance, resource utilization, and user experience, continuously improving performance based on monitoring feedback.

5.5.7 Ensure secure releases and deployments by implementing safe deployment practices, such as secure transport protocols (HTTPS, SFTP) and encrypting sensitive data, using security and reliability scanning tools like SonarQube or OWASP ZAP to identify and address vulnerabilities..

5.5.8 Employ high-level capture and logging mechanisms in the production environment to facilitate the analysis and resolution of errors and issues through appropriate error handling techniques.

5.5.9 Conduct user acceptance testing prior to software deployment, involving end users in testing and providing feedback to ensure a smooth user experience.

5.5.10 Provide comprehensive documentation for software package deployment procedures, including configuration settings and troubleshooting guides. Offer user support channels, such as email or chat, to assist end users in case of issues.

5.5.11 Develop extensive documentation and training materials to support end users, administrators, and developers, including user guides, API documentation, troubleshooting manuals, and tutorials to ensure users have the necessary resources for effective product usage and maintenance.

5.5.12 Ensure that the software system complies with regulatory, industrial standards, data privacy regulations, and organizational policies, which may involve conducting regular audits and implementing security and reliability measures while following best practices.

5.5.13 Establish a dedicated customer support team accessible through multiple channels, including phone, email, chat, and social media, to provide prompt and effective responses. Ensure that support representatives are well-trained in delivering accurate and empathetic assistance.

5.5.14 Create a comprehensive knowledge base or FAQs, including educational videos or user guides, to help customers find answers to common problems, ensuring that self-help materials are easily accessible from the company's website or support portal.

5.5.15 Implement a ticketing system for complaints that allows customers to submit their issues or inquiries and track their progress, helping the company organize and prioritize support requests, ensuring all customer inquiries are handled efficiently and timely

5.5.16 Provide proactive support for potential issues by communicating with customers before they need to contact support, achieved through proactive account monitoring, analysis, or regular communication to provide necessary updates or assistance.

5.5.17 Regularly gather feedback from customers to identify areas for improvement in support processes through surveys, social media monitoring, or analyzing customer interactions to understand weaknesses and make necessary adjustments to enhance the support experience.

5.5.18 Establish clear escalation procedures for cases that cannot be resolved immediately, ensuring that complex issues are directed to specialized support teams or higher-level representatives for better assistance.

5.5.19 Create internal communication channels and knowledge-sharing platforms to enable support representatives to collaborate and learn from each other's experiences to collectively resolve challenging customer issues, facilitated by team meetings, discussion forums, or internal documentation.

5.5.20 Provide ongoing training and development opportunities for support representatives to enhance their knowledge of software products, customer service skills, and problem-solving capabilities, enabling them to deliver more effective support and stay updated on any product or policy changes.

5.5.21 Provide multilingual support to enhance the support experience and customer satisfaction.

6. Software Governance

Software governance refers to the processes, policies, and controls that are adopted to ensure that activities related to software development and management are consistent with the beneficiary's goals, local and international standards, and regulatory requirements. Software governance includes various aspects of software development, deployment, and maintenance, including project management, decision-making, risk management, security, and regulatory compliance. An effective software governance framework typically includes:

6.1 Policies and Procedures:

Establishing manuals and standards for software development, implementation, and maintenance, including software selection, architecture, design, programming, testing, and documentation.

6.2 Roles and Responsibilities:

Defining and assign roles and responsibilities to ensure proper oversight and accountability over the entire software development phase, as this includes roles such as software engineers, project managers, developers, testers, and operating system administrators.

6.3 Decision-making Processes:

Defining how software decisions are made, including the process of approving software changes, prioritizing projects, and allocating resources, to help ensure that decisions are based on valid information and consistent with business objectives.

6.4 Risk Management:

Identifying and evaluating potential risks associated with software development and deployment, and implement strategies to mitigate those risks, including addressing security vulnerabilities, maintaining data privacy, and ensuring compliance with regulatory requirements.

6.5 Software Quality Assurance:

Implementing processes and tools to monitor and evaluate software quality throughout its lifecycle. This includes continuous testing, source code review, and adherence to programming standards and best practices.

6.6 Change Management:

Establishing procedures for managing software changes, including collecting change requirements, analyzing the impact of the change, testing, and deployment, so that change management ensures that software changes are made in a planned, tested, and effective manner.

6.7 Performance Monitoring and Reporting:

Regularly monitor software performance, measure it against pre-defined metrics and report results to stakeholders to help identify areas for improvement and support decision-making processes.


CST
هيئة الاتصالات والفضاء والتقنية
Communications, Space &
Technology Commission