SOFTWARE ENGINEERING BEST PRACTICES FOR DEVELOPING SECURE AND RELIABLE APPLICATIONS

Cristian PRODIUS

Department of Software Engineering and Automation, gr FAF-223, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Republic of Moldova

Corresponding author: Cristian PRODIUS, email: cristian.prodius@isa.utm.md

Abstract: *This paper discusses software engineering best practices for developing secure and reliable applications. Presenting a comprehensive overview of the software development process and discuss the importance of incorporating security and reliability into each stage of development. Also supplying guidelines for secure coding, testing, and deployment, and recommendations for ongoing maintenance and updates. The goal is to help software developers create applications that are secure, reliable, and free from vulnerabilities.*

Key Words*: best practices, guidelines, reliability, software engineering, security*

Introduction

Software engineering is an intricate process that involves creating, testing and deploying software applications. This can be particularly challenging when it comes to assuring security and dependability of the application. In this paper we'll cover best practices for developing secure yet dependable applications - essential in guaranteeing software is free from vulnerabilities and performs as expected [1].

The software development process involves several stages, such as planning, design, implementation, testing and deployment. Each stage plays an integral role in the project's success and presents unique challenges when it comes to security and reliability of the application. To address these concerns, we will discuss each stage in detail and provide guidelines for incorporating security and reliability into each one.

Guidelines for Secure Coding

Secure coding is an essential practice for creating software that's reliable, robust, and secure. Here are some guidelines on secure coding:[2]

1. Input validation: Validate all inputs received from the user, such as form data, URLs and parameters passed in API requests. Doing this helps protect against malicious code injection and other forms of attacks.
2. Avoid hardcoding credentials: Be careful when hardcoding passwords, access tokens or other sensitive information into your code. Instead, utilize environment variables or configuration files instead.
3. Use encryption: Encrypt sensitive data while it is stored and in transit, such as using HTTPS for web communication and encrypted storage for user data.
4. Protect against SQL injection: Utilize prepared statements or parameterized queries to protect against SQL injection attacks. Sanitize all user input to prevent malicious SQL statements from being executed.
5. Keep up-to-date: Make sure all libraries, frameworks and dependencies are up-to-date. Patch security vulnerabilities as soon as possible to avoid major disruption.

By adhering to these guidelines, you can help guarantee that your code is secure and resistant to attacks.

Guidelines for Testing

Software testing is essential to guarantee software products meet desired quality and performance standards. To do this, it's necessary to create a comprehensive test plan which outlines the testing strategy, scope, goals, test cases, and schedule. Test cases must be created that account for all potential scenarios and edge cases, with concise yet comprehensive descriptions. The testing process should then be executed systematically, with results recorded and any issues highlighted. Regression testing should be carried out to guarantee that changes made to the software do not break any previously tested functionality. Automated testing tools can reduce testing time and effort while increasing accuracy. Test data management, setting up a testing environment, using a defect tracking system, conducting code reviews and implementing continuous testing practices are all key elements in an efficient testing process. By adhering to these guidelines, software products will have higher quality and meet desired performance benchmarks.[3-5

Guidelines for Deployment

Deploying software is an essential process that can significantly impact its dependability and security. To guarantee a successful deployment, it's necessary to have an organized plan that outlines the strategy, objectives, timelines, as well as roles and responsibilities involved. Before being released to production, the deployment process should be thoroughly tested in a staging environment. This includes checking for performance, scalability and security vulnerabilities. Automated deployment tools can help guarantee consistency and reduce human error. Support version control of the software to guarantee that the correct version is deployed. A rollback plan should also be created in case any issues or unexpected results arise during deployment. Documentation and training should be given to stakeholders and end-users in order for them to become familiar with the changes, as well as enable them to utilize the software efficiently. Finally, post-deployment testing and monitoring should be conducted to confirm the software is functioning as expected and identify any issues which were overlooked during testing. By adhering to these guidelines, the deployment process can be smoother, more reliable, and secure - ultimately leading to a superior software product [6].

Guidelines for Ongoing Maintenance and Updates

Maintaining and updating software is essential to keep it running optimally and securely. To guarantee ongoing upkeep and updates, create a plan that includes regular monitoring, testing, and fixes. Regular monitoring helps detect and resolve any issues quickly, enabling rapid resolution. Testing also guarantees the software remains secure, dependable, and functional. Updates such as security patches should be performed regularly to keep everything running smoothly. It is also essential to have a backup plan in case any issues arise during the update process. Documentation, including change logs and version control, should be kept up-to-date in order to guarantee updates are tracked and stored correctly. Communication with stakeholders is essential to stay informed of any modifications or issues that might occur during maintenance or updates. Furthermore, planning for the end-of-life of the software - including decommissioning or replacement plans - must also be done. By following these guidelines closely, software can be kept and upgraded efficiently and effectively for better product security [7].

Common Coding Vulnerabilities and Recommendations for Addressing Them

Here, we will illustrate the most prevalent security vulnerabilities found in software applications, systems and networks. It is essential to comprehend these threats so that you can develop effective measures to safeguard against potential hazards. These vulnerabilities include:

1. Vulnerabilities in the Source Code: These vulnerabilities arise when errors or weaknesses exist in an application's source code. These could include buffer overflows, format string issues, and integer overflows.

2. Misconfigured System Components: These vulnerabilities occur due to incorrectly configured system components, such as firewalls, routers and servers. Misconfigurations can allow unintended access to sensitive information or unauthorized system access.
3. Trust Configuration: Trust configuration vulnerabilities may arise when trust relationships between systems are improperly set. This could grant unauthorized access to sensitive data and system resources. s.
4. Weak Credentialing Practices: Weak credentials can leave systems and networks vulnerable to unauthorized access. This includes using weak passwords, sharing of passwords, and failing to regularly update or rotate passwords.
5. Lack of Strong Encryption: Without proper encryption, sensitive data can be intercepted or accessed by unauthorized parties.
6. Insider Threat: Insider threats arise when trusted individuals with access to sensitive information negligently or intentionally misuse or reveal that data.
7. Psychological Vulnerability: Psychological vulnerabilities occur when individuals are vulnerable to social engineering attacks such as phishing or pretexting, which may trick them into disclosing sensitive information or taking actions which compromise security.
8. Inadequate Authentication: Failure to implement and maintain authentication mechanisms correctly can allow unauthorized access to sensitive data or systems.
9. Injection Flaws: Injection flaws occur when malicious code is accidentally introduced into an application through input fields like forms or search bars that are not properly verified.
10. Sensitive Data Exposure: Sensitive data exposure occurs when sensitive information, such as passwords, credit card numbers or health records is stored or transmitted without adequate protection.
11. Insufficient Monitoring and Logs: When security events and logs are not properly checked or recorded, it can be difficult to detect and address potential security incidents.
12. Shared Tenancy Vulnerabilities: Shared tenancy vulnerabilities arise when multiple users or tenants utilize the same system or network infrastructure. This could grant unauthorized access to data and resources.

Briefly, this provides a concise overview of the most common security vulnerabilities found in software applications, systems and networks. Recognizing these threats is essential for creating effective security measures that guard against potential risks. It emphasizes the significance of adopting sound security practices such as strong credentialing, encryption and authentication methods as well as regularly checking systems and logs to detect and address potential security incidents [8].

The Software Development Process

When creating software, several stages must be completed to guarantee the final product meets desired requirements and specifications. This iterative cycle promotes continuous feedback and improvement throughout the software development process. These steps include:

1. Requirements Analysis: This step involves discovering and documenting the requirements for the software application. It includes understanding user needs, recognizing any constraints, as well as deciding the scope of the project.
2. Planning: In this stage, a project plan is created based on the requirements outlined earlier. It includes timelines, budget estimates and resources needed for success.
3. Software Design: The software design stage requires creating a comprehensive plan for the application. This includes specifying the architecture, selecting the technology stack, creating a data model and designing the user interface.
4. Software Development: This stage involves executing the design by writing code using the chosen technology stack. Additionally, debugging, testing, and integration of different modules take place during this period.

5. Testing: At this stage, the software application is thoroughly tested to guarantee it meets its requirements and specifications. This includes functional testing, performance testing, and security testing.
6. Deployment: At this stage, the software application is released to production. This involves installing, configuring and releasing it.
7. Maintenance stag: Maintenance Stage, also known as Ongoing Support and Maintenance, entails providing ongoing assistance and upkeep of a software application. This includes correcting bugs, updating programs, and providing technical assistance to end-users.

Thus, the various steps involved in designing, developing, testing, and deploying software applications emphasize the necessity of an iterative approach that incorporates constant feedback and improvement to guarantee that the final product meets desired requirements and specifications.

Conclusions

Software engineering best practices are essential for creating secure and dependable applications. By incorporating security and reliability into every stage of the software development process, software developers can produce applications free from vulnerabilities that perform as expected.

References

1. HOWARD M. and LEBLANC D., Writing Secure Code, 2nd Edition, Microsoft Press, 2002. [online]. [accessed 05.03.2023].
2. MCGRAW G., Software Security: Building [online]. [accessed 05.03.2023].
3. MCCONELL S., "Code Complete: A Practical Handbook of Software Construction," Microsoft Press, 1993. [online]. [accessed 05.03.2023].
4. BOEHM B., "Software Engineering Economics," Prentice Hall, 1981. [online]. [accessed 05.03.2023].
5. SOMMERVILLE I., "Software Engineering," Addison-Wesley, 2015. [online]. [accessed 02.03.2023].
6. FOWER M, "Refactoring: Improving the Design of Existing Code," Addison-Wesley, 1999. [online]. [accessed 03.03.2023].
7. MARTIN R., "Clean Code: A Handbook of Agile Software Craftsmanship," Prentice Hall, 2008. [online]. [accessed 07.03.2023].
8. PARNAS, D. L. Software Aging. In Proceedings of the 16th International Conference on Software Engineering (ICSE '94), Sorrento, Italy, May 1994 [online]. 1994, pp. 279-287. [accessed 06.03.2023]. Available from: https://doi.org/10.1109/ICSE.1994.296788