Ticket 2 SQAP

CS5213 Group B

Feb 03, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

Creating a Software Quality Assurance Plan (SQAP) for our Capstone Management System is a strategic initiative designed to guarantee the software's optimal performance, security, and user satisfaction. This document is crafted in response to the pivotal role of quality assurance, underscored by notable software failures in the industry, aiming to deliver a system that is robust, efficient, and reliable. It will detail proactive strategies for identifying and addressing potential issues early, ensuring a seamless user experience and protecting against customer dissatisfaction and security vulnerabilities. Prioritizing quality from the start is intended to cultivate customer loyalty, minimize long-term development costs, and facilitate continuous system improvement. This approach will steer the development process to ensure the Capstone Management System maintains the highest quality standards, mirroring our commitment to excellence and innovation. In general the SQAP typically includes details on the following aspects: Roles and Responsibilities: Definitions of the roles and responsibilities of team members involved in the project, especially those related to quality assurance activities. Documentation: Guidelines on the types and formats of documentation to be produced during the software development lifecycle, including requirements specifications, design documents, test plans, and user manuals. Tools and Techniques: Information on the tools and techniques that will be used for quality control and quality assurance, including software for tracking defects, testing software, and managing documentation. Process Descriptions: Descriptions of the processes and activities that will be followed to ensure quality, including development, design, testing, and review processes. Risk Management: Strategies for identifying, analyzing, and managing risks that could impact the quality of the software product.

CHAPTER

TWO

SOFTWARE REQUIRENMENTS

Software requirements play a critical role in determining software quality by defining clear, complete, and feasible functional and non-functional expectations. High-quality requirements ensure that the software meets user needs, is reliable, maintainable, and scalable. To ensure high-quality software, the requirements gathering process should involve thorough stakeholder consultation to capture all user needs, prioritization of requirements to manage scope and complexity, and iterative refinement to accommodate changing needs and technological advances. This approach ensures the development of a system that is not only aligned with users' expectations but also adaptable to future requirements, thus maintaining its relevance and utility over time.

The software requirements for a Capstone Management System aimed at improving project management and collaboration among students, faculty, mentors, and administrators. It includes detailed requirements for various functionalities such as user authentication, role-based dashboards, project proposal submissions, a project repository, team formation, communication tools, milestone tracking, feedback mechanisms, a calendar with notifications, and reporting features. These requirements are intended to ensure the system is scalable, secure, and effectively meets the needs of all stakeholders involved in capstone projects.

CHAPTER

THREE

DEVOPS

DevOps is a collection of philosophies which highlight that software is no longer able to be designed, developed, and deployed in a vacuum. It combines the principles of developers with the principles of operators and invites software development organizations to consider the practical implications of design and implementation choices early in the process of developing software. The software development lifecycle is represented as a closed loop model in which the operation and monitoring outputs of a software system drive the inputs the planning and designing phases of the development process. Modern software technologies incentivize early deployment of the core functionality of a software product, using operational feedback and customer use patterns to inform future work.

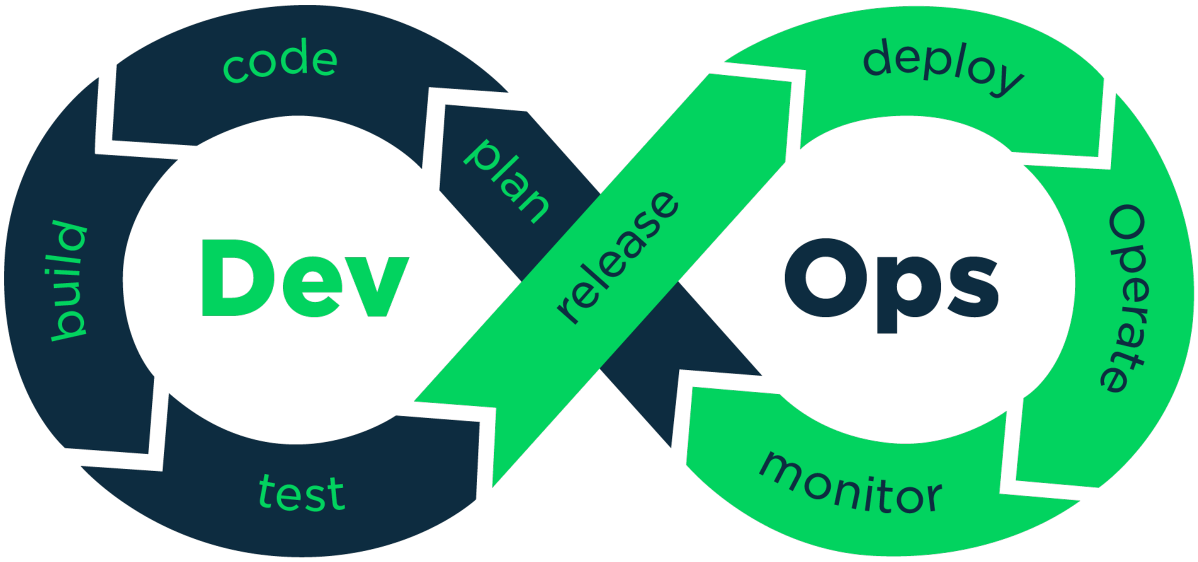

Fig. 1: The DevOps cycle. (https://wsmintl.com/blog/increase-efficiency-using-devops-manufacturing/)

The figure above shows a notional model of the DevOps cycle, which is really just the software development lifecycle with the addition of the "monitor" phase. This monitor phase is characterized by the use of introspection and feedback tools to drive the inputs of the planning and design phases of software development. This monitoring emphasis draws the operations teams closer to the development teams within the organization. The domain knowledge of IT experts can improve the design and security of software systems being built by the development teams, and the expertise of the software developers can be leveraged by the IT team to automate and simplify their repetitive tasks.

DevOps does not replace Agile methodologies for software development, rather it augments those processes with the principles and practical considerations from operations.

3.1 Tools

“To the man who only has a hammer, everything he encounters begins to look like a nail.”

—Abraham Maslow

Ensuring the quality of a software product entails using many tools which allow development teams, project management, and operational staff to effectively carry out the phases of the software development lifecycle. Without the proper tools, a software project is doomed to fail.

3.1.1 Git

Git is a version control software that has become the primary tool for managing the versioning and history of changes in software projects. It has become a ubiquitous tool with which every developer must become familiar in the course of their work. Git is a distributed version control system which stores a binary database of the changes or “diffs” in text files. These diffs are combined into commits when the developer uses the `git add` and `git commit` commands. This collection of changes is called a “repository”, the local repository is affected by the `git add` and `git commit` commands, while the remote repository which amounts to the “source of truth” for all developers on the project is updated by the `git push` command. Using these three basic commands, a developer can begin to collaborate and contribute to massive a highly distributed software development projects. Beyond this basic workflow, Git supports a variety of commands and workflows that allow for more complex management of the changes in a project. Git is a command line tool with build support for all major operating systems (Windows, Linux, and Mac OS).

Git will be used to track the history of changes to the software product. This contributes to the quality of the software by allowing developers to track and revert changes that are detrimental to the codebase.

3.1.2 GitHub

GitHub is a web application which augments the command line tool Git with additional features which were not considered in the original scope of Git. GitHub is in a family of applications which all provide similar feature sets for extending Git with various quality-of-life and project management features such as: branch protections, forking repositories, merge/pull requests, issue boards, etc. These features have become synonymous with Git, despite not being available in the tool itself. For managing larger code bases, applications like Github help facilitate the complex version histories and configurations. GitHub provides a user-friendly web interface which some developers prefer over the command line.

GitHub will be used to manage the repository, projects (sprint boards, Kanban boards, milestones, etc.), issues (bug reports), and for CI/CD and artifacts.

3.1.3 GitHub Actions

GitHub Actions is a feature of GitHub which allows teams to create workflows consisting of "actions." Actions are individual processes and operations typically related to specific components of the software development lifecycle such as compiling source code, generating documentation, performing automated tests, running static analysis tools, or generating reports. Actions are performed on virtual machines or containers run in cloud infrastructure or on-premises runners. Actions and workflows encourage development teams to capture elements of the development environment that could easily be undocumented or unmaintained and enshrine them in *configuration-as-code*.

GitHub actions will facilitate CI/CD and artifact management for builds, improving the quality of the software by virtue of releasing early and often.

3.1.4 GitHub Projects

GitHub projects will be used to manage scrum sprint boards, Kanban boards, issues, and milestones for the project. Milestones can be defined by releases at the end of each sprint and the major features the project managers and scrum masters have determined can be accomplished by the scrum teams. Issues consist of feature requests and bug reports from users and known issues reported by developers. Scrum sprint and Kanban boards are used to track work across sprints or in the Kanban style.

Kanban boards are best suited for workflows in which tasks move through a well-defined progression of statuses and ownership of tasks may change hands across the process. In scrum, a task is usually assigned start-to-finish to one person or small group of people. In a Kanban workflow, the task moves between states and potentially owners while flowing through the process. The board allows project managers to track multiple tasks simultaneously as they move through the workflow. The tasks themselves will contain descriptions and comments which inform the project manager of progress, roadblocks, and required actions to move the task forward.

GitHub projects are useful for both types of project and will improve software quality by integrating project management more tightly with software development.

3.1.5 Sphinx Document Generation

Software developers are notoriously bad at opening word processors and documenting their work. Thankfully, there are tools which exist to help make documentation more like programming. Sphinx is a document generator that can ingest documentation either directly from source code or from plaintext files and output the documentation described therein in various formats. HTML webpages and PDF documents are two possible outputs of this tool.

Using Sphinx, the documentation of the CMS project will be kept up-to-date with the code and technical manuals will be simple to write. Developers can continue to use their favorite text editors and avoid cumbersome operations like text formatting. Shifting the focus entirely to content lifts a great deal of the overhead off of the developer when generating documentation.

Document generation tools, like Sphinx, can easily be incorporated in to the CI/CD pipeline to automatically build documentation with every commit the development team makes. This offers a high potential to increase software quality by keeping one of the major by-products of a software development effort, the documentation, close to the source code.

3.2 Continuous Integration / Continuous Deployment

Two core principles of DevOps are often paired, Continuous Integration and Continuous Deployment (CI/CD).

Continuous Integration
The principle of rapidly integrating code changes, performing tests, producing documentation, and performing other software development lifecycle activities, typically by way of some automation platform. The *de facto* version control tool is Git, and so typically these actions are performed any time code is committed and *pushed* to the shared remote repository (the `git commit` and `git push` commands).

Continuous Deployment
The principle of a development pipeline which is capable of carrying out a sufficient amount of the software development process to produce all major artifacts of the software from a given commit (or set of commits, in a multirepo), and in some contexts, deploy the software to testing, staging, and even production environments, either directly or with minimal human intervention.

By embracing CI/CD, development teams will increase agility and software quality by moving the code through software development lifecycle stages more often. Deploying earlier and more often will make latent bugs more obvious and force the team to tackle staging and production environment issues while the software is still relatively simple. Often, teams leave deployment to IT or operations professionals and treat this as a handoff which leaves opportunities for quality improvement off the table.

For the CMS project, using GitHub actions to automate building, testing, and staging the CMS platform for deployment will allow it to quickly mature into a robust platform. The only feasible way to accomplish this is with a focus on CI/CD from the first sprint.

3.2.1 Artifacts

To track and configuration manage the outputs of the various processes in the development cycle for the CMS product, each major task, compiling code, running tests, generating technical documentation, should be defined as a job in GitHub actions. The artifacts of these jobs, compiled binaries, test reports, and technical documents should be specified in an upload artifact action in GitHub actions. As actions are typically run on a commit-by-commit basis, this will capture the artifacts at every commit. This behavior can be tuned on an action-by-action basis if certain artifacts change slowly and are expensive to store in this manner. The simplest approach is to start by collecting every output for every commit and trimming things down as necessary.

CHAPTER

FOUR

CONFIGURATION MANAGEMENT

Configuration Management (CM) is a key part of DevOps. It helps manage changes in a careful way to keep the software and its parts reliable and easy to track through its entire life. The Capstone Management System (CMS) will follow strict CM rules. This is to make sure everything is consistent, works well, and is of good quality when we deliver its features and services.

4.1 Software Artifacts

An artifact is a consequential byproduct emerging from the software development process, serving to delineate the software's architecture, design, and functionality. These artifacts function analogously to roadmaps, enabling software developers to methodically trace the progression of the entire software development lifecycle.

Such artifacts may manifest in various forms, including but not limited to databases, data models, printed documents, or scripts. They are instrumental in the ongoing maintenance and updating of software, providing developers with vital reference materials that facilitate the resolution of diverse issues. To ensure accessibility and organization, these artifacts are systematically documented and securely stored in a designated repository, thereby allowing software developers to retrieve them as required.

The Capstone Management System (CMS) will establish a complete Software Artifacts. This system is designed to systematically manage all software-related products during every stage of the development process. It aims to maintain the integrity and constant availability of these artifacts. This approach will strengthen the overall strength and dependability of the CMS.

4.1.1 Version Control System (VCS) Implementation:

A VCS such as Git will be employed to manage changes to the software codebase. The VCS will enable developers to commit changes, track history, revert to previous states, and branch out for feature-specific developments without affecting the main codebase.

4.1.2 Repository Management:

Tools like GitHub or GitLab will act as central repositories for hosting the VCS. These platforms will support collaborative features such as pull requests and code reviews, fostering team collaboration and quality assurance through peer feedback.

4.1.3 Artifact Storage and Distribution:

Artifacts produced from the build process, such as binaries and libraries, will be stored and versioned in an artifact repository (e.g., JFrog Artifactory or Nexus Repository). This allows for secure storage, categorization, and distribution of software artifacts.

4.1.4 Automated Testing and Integration:

Integration with Continuous Integration (CI) tools (such as Jenkins, CircleCI, or GitHub Actions) will be set up to automate the testing and building processes. Upon each commit, the CI tool will run a suite of automated tests to validate the integrity of the code and ensure that new changes do not introduce regressions.

4.2 Documentation

Technical documentation in software engineering refers to all the written materials and documents related to the development of a software product. This type of documentation is necessary for every software development project, regardless of whether it's undertaken by a small team or a large corporation. Different kinds of documents are produced at various stages of the software development lifecycle (SDLC). The purpose of this documentation is to describe the product's functionality, consolidate information pertinent to the project, and facilitate communication about important issues among stakeholders and developers.

Software documentation plays a crucial role as it aids users in understanding how to operate the software. It also provides essential technical information to developers and other technical stakeholders. Additionally, it ensures that the software development process is both consistent and repeatable. Well-crafted and comprehensive documentation not only improves the overall quality of the software but also enhances the user experience. The Capstone Management System (CMS) will follow a detailed and careful method for its documentation. It will make sure that every part of the system and its processes are clearly explained and easy to find. This approach to documentation is very important to keep the system user-friendly, easy to support, and consistent.

The documentation suite for the CMS is designed to:

- Clearly explain system operations and processes to all involved parties.
- Give comprehensive instructions to ensure easy usage for end-users.
- Promote system maintainability and scalability with precise technical documentation.
- Keep references current, in line with any changes or updates to the system.

4.2.1 Release Documentation

In the context of the Capstone Management System (CMS), Release Documentation is a critical component that communicates the specifics of each software update or release to the system's stakeholders, which includes students, faculty, administrators, and company representatives. This documentation facilitates transparency and informs the users about new features, improvements, bug fixes, and any changes they might experience.

The fundamental objectives of Release Documentation are :

- Document the evolution of the CMS, providing a clear history of its development and updates.
- Communicate changes effectively to users and stakeholders to prepare them for new versions.
- Serve as an official record of the features, fixes, and updates included in each release.

The Release Documentation for each CMS update will typically include:

- Version Information: A unique identifier for the release, usually following semantic versioning (e.g., version 1.2.3), which facilitates easy reference.
- Release Date: The date when the version was released to the users, which helps in tracking and support.
- New Features and Enhancements: Detailed descriptions of new functionalities, improvements to existing features, and any user interface updates. This might include screenshots or visuals for more significant updates.
- Bug Fixes: A list of bugs addressed and resolved in the new release, providing acknowledgment of issues reported by users and the steps taken to address them.
- Known Issues and Limitations: Transparency about any known limitations or issues that remain in the release, providing users with a realistic expectation of system performance.
- Migration Instructions: For users upgrading from an older version, step-by-step instructions or scripts to facilitate a smooth transition, including any configuration changes or steps to update user data.
- Deprecations: Information on any features or components that are no longer supported or will be removed in future releases.
- Contributors: Acknowledgment of the team members and contributors who participated in the release, which can help in humanizing the development process and providing credit where it's due.
- Technical Notes: Any additional technical details that would be relevant for the IT staff managing the CMS, such as changes in dependencies, environment configurations, or API modifications.

4.2.2 User Documentation:

User documentation is an important component of the Capstone Management System , designed to guide various stakeholders through the efficient and effective use of the system. This documentation is customized to meet the requirements of the users, ensuring that students, faculty, administrators, and company representatives can navigate and utilize the CMS with ease and confidence. The primary goals of the User Documentation for the CMS are to:

- Facilitate a seamless user onboarding experience.
- Provide clear instructions and support for common tasks and procedures within the CMS.
- Enhance user self-sufficiency by offering answers to frequently asked questions and troubleshooting common issues.
- Ensure consistent use of the CMS features and tools in line with best practices.

The User Documentation is composed of several key elements, each serving a specific purpose:

- Getting Started Guides: Introductory materials that help new users familiarize themselves with the CMS interface and its basic functionalities. These guides include step-by-step instructions for initial setup, such as account creation and login processes.
- Feature Overviews: Detailed descriptions of each feature within the CMS, explaining its purpose, how to access it, and its use cases. These overviews provide a comprehensive understanding of the system's capabilities.
- Tutorials and How-To Guides: Step-by-step instructions that walk users through various processes, such as submitting project proposals, forming teams, utilizing communication tools, and tracking project progress. These guides may include annotated screenshots, workflow diagrams, and video tutorials for complex tasks.
- Accessibility Information: Guidelines ensuring that all users, regardless of ability, can access and use the CMS. This includes information on keyboard shortcuts, screen reader compatibility, and other accessibility features.

CHAPTER

FIVE

INFRASTRUCTURE

The choice of infrastructure for the Capstone Management System plays a crucial role in the overall Software Quality Assurance (SQA) plan. Ensuring that the system is reliable, scalable, secure, and maintainable is fundamental to achieving high software quality. This section outlines considerations and strategies for infrastructure choice in relation to the SQA plan for the Capstone Management System.

5.1 Infrastructure Considerations for SQA

1. Scalability: The infrastructure must support scaling to accommodate varying loads, especially during peak usage periods. Scalability ensures that the system can handle growth in user numbers and data volume without degradation in performance or user experience.
2. Reliability and Availability: High availability configurations and reliable infrastructure components minimize downtime and ensure that the system is accessible whenever needed. This involves choosing robust server hardware or reliable cloud service providers, along with implementing redundancy and failover strategies.
3. Performance: The infrastructure should support fast response times and efficient processing capabilities to meet performance requirements. This includes adequate computing resources, optimized databases, and effective caching strategies.
4. Security: Protecting sensitive data and ensuring user privacy requires an infrastructure that supports strong security measures. This includes firewalls, encryption, secure access controls, and regular security updates to mitigate vulnerabilities.
5. Maintainability and Monitoring: Infrastructure that is easy to update, monitor, and maintain supports ongoing quality assurance efforts. This involves using tools for continuous integration/continuous deployment (CI/CD), automated backups, and real-time monitoring to identify and resolve issues promptly.
6. Compliance: For systems that must adhere to specific regulations, the infrastructure must support compliance requirements, such as data protection standards and audit capabilities.

5.2 Infrastructure Choices for the Capstone Management System

For a student project focused on developing a Capstone Management System, the choice of infrastructure is pivotal for balancing cost, usability, scalability, and maintainability, especially when prioritizing open-source or free resources. The infrastructure should support the system's needs, including web hosting, database management, version control, continuous integration/continuous deployment (CI/CD), and project collaboration tools.

1. Web Server and Hosting - GitHub Pages offers free hosting solutions that are sufficient for projects with straightforward, client-side processing needs. It is user-friendly and integrate well with version control.

2. Database Management - PostgreSQL is an open-source, robust, and feature-rich database system. For small to medium-sized applications, PostgreSQL offers extensive documentation and a strong community. It can be hosted locally for development and testing, with deployment on platforms like Docker Hub for production.
3. Version Control and Collaboration - As the standard for version control, GitHub offers unlimited private repositories for free, supporting collaboration, issue tracking, and code review. GitHub's Actions and Workflows also provide CI/CD capabilities, automating tests, and deployment processes directly from the repository.
4. Development Environment - Visual Studio Code is a free, open-source code editor that supports a wide range of programming languages and frameworks with a vast library of extensions. It's lightweight, powerful, and includes integrated Git control, simplifying code development and collaboration.
5. Containerization - Docker allows developers to package applications into containers, standardizing environments across development, testing, and production. This ensures consistency and simplifies deployment and scaling. Docker's community edition is free and widely supported.
6. Monitoring - Prometheus is an open-source system monitoring and alerting toolkit with a focus on reliability and simplicity. Prometheus is designed for reliability, to be the system you go to during an outage to allow you to quickly diagnose problems. It can be used to collect and process metrics from various sources, including microservices, databases, and infrastructure components. It uses dimensional databse model.
7. Logging - ELK Stack (Elasticsearch, Logstash, and Kibana) is an open-source collection of tools that work together to provide an end-to-end log analysis solution. Elasticsearch acts as a search and analytics engine. Logstash is responsible for processing and aggregating the logs, while Kibana is used for visualizing the data stored in Elasticsearch. This stack can handle massive amounts of logs, making it suitable for projects of any size. For smaller projects or those just starting, Elastic offers a free tier of their managed Elasticsearch service, which can simplify setup and maintenance.

5.3 Integrating Infrastructure into SQAP

Managing infrastructure-related quality aspects involves a coordinated effort across multiple roles within the development team, each with specific responsibilities to ensure the application meets performance, security, and compliance standards. Here is an outline of these responsibilities:

DevOps Engineers

- CI/CD Pipeline Management: Set up and maintain Continuous Integration and Continuous Deployment pipelines using tools like GitHub Actions to automate testing, building, and deployment processes.
- Docker Container Management: Ensure consistent application environments across development, testing, and production through Docker, managing container orchestration to optimize resource use and scalability.

Database Administrators

- Database Performance Tuning: Regularly monitor and optimize the performance of PostgreSQL databases, ensuring efficient data storage, retrieval, and integrity.
- Data Security and Compliance: Implement and maintain database security measures, conduct vulnerability assessments, and ensure compliance with data protection regulations.

Security Specialists

- Security Audits: Conduct regular security assessments, including code reviews and penetration testing, to identify and remediate vulnerabilities.
- Compliance Checks: Ensure the application and its infrastructure comply with relevant industry standards and regulations, preparing for audits by external bodies if required.

Quality Assurance (QA) Engineers

- Performance Testing: Design, implement, and execute performance tests using scenarios that simulate real-world usage to identify bottlenecks and scalability issues.
- Monitoring and Logging: Utilize the tool Prometheus for real-time monitoring and logging to detect and diagnose issues early.

Developers

- Code Quality and Security: Follow best practices for coding, utilizing Visual Studio Code extensions for code analysis and security checks, ensuring high-quality, secure code commits.
- Documentation: Maintain clear and comprehensive documentation hosted on GitHub Pages, including API documentation, user manuals, and deployment guides, ensuring they are up-to-date and comply with project standards.

Project Managers

- Coordination and Oversight: Oversee the project's progress, ensuring that all team members fulfill their responsibilities regarding performance, security, and compliance.
- Risk Management: Identify potential infrastructure-related risks and coordinate with the respective team members to mitigate them. By clearly defining these responsibilities, the team can ensure a high-quality, secure, and compliant infrastructure for the Capstone Management System, ultimately contributing to the project's success and reliability.

CHAPTER

SIX

DOCUMENTATION AND CODING STANDARDS

When developing and maintaining software, it is important to keep your documentation and coding standards upheld, since this will both help the stakeholders of your software and the development team. In this section we will go over documentation standards, which will be pertinant to how this team and future teams should communicate with the stakeholders of this software. The coding practices and standards will benefit the development team and whomever maintains this codebase in the future. Below details how the standard documentation for the customers, stakeholders, and developers will be designed and implemented.

6.1 Release

Keeping our customers and stakeholders informed on the product, it is important to have clear and effective communication of all the features and maintenance performed on the software. To this end, semantic versioning should be the template used for this project when release notes and version control are utilized. This system is already widely adopted across most comercially developed products, giving stakeholders an intuition on the progression and maintenance performed on the product.

Publishing a release schedule is an important task for clear communication with the stakeholders. For this project, a consistent release schedule or periodic release schedule could be adapted, but with different benefits and drawbacks. For the small project, a periodic/irregular release schedule may suffice, since this would allow the development/maintenance team to implement patches or features at the customer's request. If this software scales in usage beyond a couple of customers, a more regular release schedule may be needed to ensure continuous maintenance and additional features for a variety of customer needs.

There are also a number of different stylistic choices that can be made within the document that can be used. When release documentation is being developed, the team may construct a template that will be used to standardize all the release documents throughout the service life of this software. The release notes will contain the following information:

- Semantic enumeration (discussed previously)
- Brief introduction and overview of the changes that were implemented
- Changes from the previous releases
- Users impacted (this is more important when if the software experiences a scaling of users)
- Ongoing maintenance/issues/features that will be addressed in the future

The sections presented above should be clearly divided within the document, with additional divisions created between the features and bug fixes. All the release notes should be stored together and users should be able to clearly access all the documents that have been produced by the team. Release notes may utilize a numerous of different media options, from text to images, diagrams, and even videos. A text version of the release notes is common practice, and this is often the groundwork of the utilization of other media types. For the scale of the project, it may be best to utilize text embedded with images or diagrams as needed. These release notes should be developed by the whole development team, depending on the scope of the bug or feature implemented. Bug fixes should either be enumerated or given a brief description by the developer(s) that patched the problem. Minor features or quality-of-life patches should also be described at this level. Major bugs patches and feature implementation should be discussed by the Product Manager (PM) or management team considering the scope that these bugs/features have on the application as a whole.

6.2 User Documents/User Guide

User documents help customers get aquainted with the full potential of the software. This provides the user a brief tutorial of the features of the software, installation instructions (if necessary), and a contact to the development team. Below enumeratues the contents that should be included in the user documents.

1. A brief introduction to the software. This aims to help the user understand what the software was designed to do, the core features that are contained within the software, and support given to the software. By describing the core features of the software briefly along with the intended audeince provides the customer with information on whether the software will cover their needs. Support information in the sense of a brief description may be a timestamp of the past update or two. This informs the customer if the software is being maintained, which should intice the customer to utilize this product.
2. A tutorial of the core features of the application. By providing a tutorial of the core features, this aims to not overwhelm the customers and potential clients, and introduce them to the layout of the application. This enables the customer to utilize the software's core features, and empower them to dig deeper towards the other features and tools implemented in the application.
3. An in-depth description and instructions for utilization on all the features on the application. This portion of the docuent will need to be continually updated as patches and updates are deployed throughout the lifecycle of the applicaiton. This gives the user a full picture of all the features that are available within the software, and giving stakeholders an idea of all the different featues that are given within the software, giving the potential to attract potential clients. This also allows users to understand and use the application to its full potential, and give the users ideas of other features that they would want this application to encorporate. Within this section will include a brief description on how to access and use these features.
4. Support information and contact information. This section will include a link to where the release notes will be stored, and contact information to the support team. This allows the user access to the support and maintenance given to the application, and a means to contact the support team if further maintenance is needed.

6.3 Coding Standards

Implementing common coding standards is important to the maintenance of the application. Developers using a common syntax and documentation standard allows for maintenance to be performed across the platform despite whether the original developers are still on the program. Additionally, this common practice will ease integration of new features since it will be clear how the application is connected. This coding standard sets limits on a number of practices, from code documentation, imports, class definitions, and more. With the project being compliant with this standard, it will help both the developers on the team and any new developers that may join the team in the future. Below we enumerate which style guide this project will adhere to based on the different languages used throughout the technology stack.

- Python: Google python coding style guide
- SQL: Mozilla Data Documentation SQL style guide
- JavaScript: Google JavaScript style guide
- HTML/CSS: Google HTML/CSS style guide

CHAPTER

SEVEN

CHANGE CONTROL PROCESS

The change control process is used to ensure that all changes made to the configuration items are recorded, evaluated, approved, and tracked. The change control process is critical for maintaining the stability and integrity of the system, preventing unauthorized changes, and ensuring that all modifications meet the necessary requirements before implementation. A structured change control process is essential to manage and implement changes effectively without disrupting the project timeline.

7.1 Stages of Change Contol

1. **Request for Change** In the context of 4-Week sprint development plan, changes can be raised at any stage of the sprint. It often starts with identifying a gap, a requirement not met, or an opportunity for improvement. Team members or stakeholders submit a change request, which should be documented clearly, preferably using a template that includes the change's description, reason, impact, benefits, test cases and required resources.

- Upon reporting, issues are classified by the software development team into one of three categories: defect, enhancement, or user error. Use of GitHub is recommended for opening issues and to assign specific issues to UI developers, Database developers, or testing teams based on the nature of the problem. Github features 'labels' and 'milestones' can be used to categorize issues (e.g., bug, enhancement, urgent) and milestones to group issues into specific targets or sprints.

2. **Identify defects and enhancements User Feedback** - User feedback is an invaluable source of insights for identifying both defects and enhancements in software. It provides direct input from those who interact with the product daily, offering a unique perspective that internal testing might not always capture. Enhancements are identified through stakeholder feedback, user suggestions, or team insights, focusing on adding new features or improving existing ones.

- **Beta Testing/Feedback Loops** - Beta testing, where features are released to a subset of users (other sprint teams) before a wider rollout, establishes an effective feedback loop. This approach allows developers to gather user impressions and identify any issues or potential improvements from a real-world perspective. Encouraging beta testers to share their experiences, problems, and suggestions can uncover defects that slipped past initial testing phases and highlight opportunities for enhancements that might significantly improve user satisfaction.

3. **Change Request Review** A designated developer/team lead reviews the request, considering its necessity, impact on the timeline, and resource requirements.Based on the review, the change can either be approved, rejected, or require more information. The decision should be communicated to all relevant stakeholders. If approved, the change moves to the planning phase where the team plans the necessary software changes, including performance upgrades, new technologies incorporation, or vulnerability patches.

4. **Implement Change** The responsible team implements the change, adhering to the project's coding standards and development practices. Below are the list of development practices of each team. Implementing a change control process on GitHub involves utilizing its robust features like branches, pull requests.For every change, whether a bug fix or feature enhancement, create a new branch from the development branch. Once the change

is ready for review, open a Pull Request against the development branch. The Pull Request description should link to the related issue(s) by mentioning the issue number.

5. **Review and Close Change** After the change has been implemented, it's important to review its impact. This includes ensuring that the change has achieved its intended outcomes and identifying any unforeseen consequences. Lessons learned during this stage can be valuable for managing future changes. Once the changes are thoroughly tested to ensure they meet the requirements and do not introduce new issues and has been successfully integrated by merging it into the development branch in Github and all documentation has been updated, the change process can be officially closed.

7.2 Team Roles in Change Control

- UI/UX Team: Submits change requests for design improvements or fixes based on user feedback.
- Database Team: Raises changes for database schema modifications or optimizations.
- Full-Stack Developers/Team Leads: Oversee the integration of front-end and back-end changes, ensuring they align with the system's overall architecture.
- Testers: Identify defects during testing phases and may suggest enhancements for better usability or performance.

Each team should be involved in the review process for changes affecting their work, using GitHub's Pull Request (PR) feature for code changes and the review process to ensure quality and alignment with project objectives.

7.3 Applying Change Control in Development Plan

Allocating time for change control involves planning for review, approval, and implementation within the sprint's timeline. In a 4-week sprint, it's advisable to allocate time at the end of each week or phase for reviewing and implementing approved changes. For instance, dedicating a portion of Week 2 and Week 3 for incorporating changes raised during the previous weeks helps manage the workload without disrupting the sprint's progress significantly.

7.4 Time Allocation for Release Changes:

- Minor Changes/Bug Fixes: Allocate 1-2 days for implementing and testing minor changes or bug fixes. This allows for quick adjustments without significantly impacting the sprint timeline.
- Major Feature Changes: For significant modifications or new features, allocate up to a week, ensuring there's enough time for implementation, testing, and integration. These should be planned to start at the beginning of a sprint cycle where possible.

CHAPTER

EIGHT

VERIFICATION AND VALIDATION

8.1 Introduction

In the ever-evolving landscape of software development, the pivotal role of testing cannot be overstated. It serves as the bedrock, ensuring the reliability, functionality, and security of the final product. This multifaceted facet involves various testing methodologies, each tailored to specific focuses and objectives. From functionality and usability testing to performance, security, and beyond, the diverse array of testing types collectively shapes a comprehensive toolkit for quality assurance. This exploration delves into the nuances of these testing types, unraveling their significance and understanding their integral role in delivering software products that not only meet but exceed user expectations.

Concurrently, the creation of a well-crafted Software Testing Plan (STP) emerges as a linchpin in the trajectory of a software development project. The STP serves as a meticulous roadmap, delineating the testing team's strategies, goals, and scope. It acts as a guiding beacon, ensuring that all software components undergo thorough testing before advancing to the release phase. This introduction unfolds the significance of an efficient STP, delving into its various components and elucidating how it aligns with project objectives to ensure a systematically executed and effective testing process. The harmonious interplay between diverse testing methodologies and a thoughtfully crafted STP forms an unassailable framework, directing development teams toward the triumphant delivery of software solutions in today's dynamic and demanding technological landscape.

8.2 Types of Testing

8.2.1 Functionality Testing

Functionality testing is fundamental to a robust quality assurance strategy, ensuring that the Capstone Management System aligns with specified requirements. This category encompasses various testing types, each addressing different facets of the system's functionality.

Unit Testing

Unit testing, a fundamental practice in software development, involves the examination of individual units or components in isolation. In the context of the Capstone Management System, this entails subjecting each module or function to rigorous testing to ensure its independent and correct operation.

Integration Testing

Integration testing takes a step further, validating the interaction and collaboration between integrated units. Given the complex architecture of the Capstone Management System, seamless integration of diverse modules is pivotal for optimal performance.

System Testing

System testing adopts a holistic approach, evaluating the complete system's functionality. This phase is essential for validating end-to-end processes within the Capstone Management System and ensuring a cohesive user experience.

Regression Testing

The iterative nature of software development necessitates regression testing, where existing functionalities are rigorously re-evaluated to ensure that new changes do not introduce unintended consequences. As the Capstone Management System evolves, regression testing becomes a linchpin for maintaining system stability.

Acceptance Testing

Acceptance testing serves as the final frontier before stakeholder approval. In the realm of the Capstone Management System, this involves a comprehensive evaluation to confirm that the system effectively manages capstone projects and aligns with user expectations.

Usability Testing

Usability testing occupies a critical space in the quality assurance landscape, focusing on ensuring that the Capstone Management System is not just functional but also user-friendly. This involves assessments of navigation, accessibility, and overall user experience for students, faculty, and administrators.

Performance Testing

Performance testing evaluates the system's responsiveness, speed, and scalability. Given the varied user base and potential data loads associated with the Capstone Management System, performance testing is instrumental in guaranteeing the platform's efficiency and reliability.

Security Testing

Security testing is paramount, especially when dealing with a platform like the Capstone Management System that may contain confidential project details. This type of testing identifies vulnerabilities and weaknesses, safeguarding sensitive data from unauthorized access.

Smoke Testing

As a preliminary test, smoke testing provides a quick assessment of the basic functionalities of the system. It acts as a gatekeeper, determining whether further, more in-depth testing is warranted.

8.2.2 Other Testing Types

End-to-End Testing

End-to-end testing evaluates the entire workflow of the Capstone Management System, ensuring seamless integration and functionality across all components.

Sanity Testing

Sanity testing offers a quick check of specific functionalities after changes or bug fixes, ensuring that the system remains stable.

White-box Testing

White-box testing delves into the internal logic and structure of the software, scrutinizing the code to ensure integrity and adherence to coding standards.

Accessibility Testing

Accessibility testing ensures that the Capstone Management System is accessible to individuals with disabilities, aligning with principles of inclusivity.

Compatibility Testing

Compatibility testing guarantees that the system functions correctly across different browsers, devices, and operating systems, enhancing the user experience.

Test Automation

Test automation involves leveraging tools to automate repetitive testing tasks, improving efficiency, and ensuring accuracy in the testing process.

8.2.3 Non-Functional Testing

Non-functional testing shifts the focus from specific functionalities to broader aspects such as performance, security, and usability, contributing to a more comprehensive evaluation.

8.2.4 Other Testing Approaches

Gray-box Testing

Gray-box testing strikes a balance between white-box and black-box testing, providing a more nuanced perspective on system functionality.

Agile Testing

Aligned with the Agile development methodology, Agile testing emphasizes continuous testing throughout the development process, ensuring adaptability and responsiveness to changing requirements.

User Acceptance Testing

User acceptance testing involves end-users validating that the Capstone Management System meets their needs and expectations, providing valuable feedback.

Age Testing

Age testing assesses how the system performs over time, considering potential issues that may arise with prolonged use.

Software Beta Testing

Software beta testing involves releasing a pre-release version to a select group of users, allowing for feedback collection and refinement before the official launch.

Exploratory Testing

Exploratory testing, with its focus on simultaneous learning, test design, and execution, proves invaluable for uncovering unexpected issues in the Capstone Management System.

8.3 Software Testing Levels

8.3.1 Unit Testing

Unit testing, as the foundational level, focuses on the evaluation of individual components, ensuring their correctness and functionality.

8.3.2 Integration Testing

Integration testing validates the interactions between integrated components, ensuring seamless collaboration.

8.3.3 System Testing

System testing evaluates the complete and integrated software system, ensuring that all components work cohesively.

8.3.4 Acceptance Testing

Acceptance testing, at the highest level, assesses the software's compliance with business requirements, gaining stakeholder approval.

8.4 Continuous Monitoring and Feedback Loops

The testing phase is not a one-time event but a continuous process. Implementing continuous monitoring mechanisms ensures that any deviations or anomalies are promptly identified and addressed. Incorporating feedback loops from various stakeholders, including users, during different testing stages allows for agile adjustments. This iterative feedback loop fosters a dynamic testing environment, aligning the development process with evolving requirements and user expectations.

8.5 Continuous Improvement

Continuous improvement is foundational to maintaining software quality over time. Regular updates to the testing plan, guided by feedback, evolving requirements, and emerging technologies, are imperative for the long-term success of the Capstone Management System.

8.6 Crafting an Efficient Software Test Plan

In the dynamic realm of software development, the creation of a robust test plan is paramount for ensuring the reliability, functionality, and security of the final product. A well-defined test plan not only guides the testing team through their strategies and objectives but also serves as a roadmap for comprehensive testing. In this context, we will explore the essential steps in creating an efficient test plan, drawing insights from industry best practices and incorporating a nuanced perspective based on practical experience.

8.6.1 Real-world Application

To provide a tangible application context, let's introduce a hypothetical scenario. Consider a situation where the Capstone Management System is on the verge of a major update. In this scenario, various testing methodologies discussed, including regression testing and usability testing, are systematically employed. The STP acts as a guiding document, helping the testing team navigate through intricate processes. The risk management strategies prove valuable when unforeseen challenges arise during testing, showcasing the practical utility of a well-defined testing framework.

8.6.2 Define the Release Scope

The foundational step in crafting an efficient test plan is defining the release scope. This involves a meticulous examination of the project's objectives, features, and functionalities. Understanding the scope sets the boundaries for testing, ensuring that every critical aspect is addressed. For the Capstone Management System, this would include a comprehensive assessment of the platform's capabilities, user interactions, and the integration of various modules.

8.6.3 Schedule Timelines

Timelines are the backbone of any successful project, and the creation of a test plan is no exception. Scheduling timelines involves a realistic estimation of the time required for each testing phase, aligning with the overall project schedule. This step ensures that testing activities are seamlessly integrated into the development lifecycle, preventing bottlenecks and delays. For the Capstone Management System, a well-defined timeline would include milestones for functionality testing, performance testing, and security testing, among others.

8.6.4 Define Test Objectives

Clarity in objectives is essential for a focused and effective testing process. Define test objectives by outlining what needs to be achieved through each testing phase. This could involve validating specific functionalities, ensuring system stability, or uncovering potential security vulnerabilities. In the case of the Capstone Management System, test objectives might include validating the user authentication system, assessing the responsiveness of the dashboard, and verifying the security measures in place.

8.6.5 Determine Test Deliverables

Test deliverables are the tangible outcomes of the testing process. This involves creating detailed test cases, test scripts, and reports that provide a comprehensive view of the testing activities. Determining test deliverables ensures that the testing team produces documentation that is not only valuable for their internal processes but also for stakeholders and developers. For the Capstone Management System, test deliverables might include detailed test cases for functionality, usability reports, and performance test results.

8.6.6 Design the Test Strategy

Designing the test strategy involves outlining the overall approach to testing. This includes identifying the types of testing to be conducted, the testing environment, and the tools and resources required. The test strategy is a crucial aspect of the test plan, guiding the testing team on how they will achieve the defined objectives. In the context of the Capstone Management System, the test strategy might involve a combination of manual and automated testing, focusing on usability, security, and performance aspects.

8.6.7 Plan Test Environment and Test Data

The success of testing often hinges on the availability of a suitable test environment and relevant test data. Planning the test environment involves creating a replica of the production environment, ensuring that testing conditions mirror real-world scenarios. In the case of the Capstone Management System, this might include configuring servers, databases, and network settings to mimic the actual deployment environment. Additionally, planning for test data involves ensuring that the system is exposed to a variety of inputs, reflecting different usage scenarios.

Deferring to Later: Standardization of Software Test Plans and Their Usage

In the realm of Software Test Plans (STPs), standardization is a critical factor in ensuring consistency and efficiency across projects. A standardized STP establishes a common framework for testing processes, making it easier for teams to collaborate, share insights, and maintain a high level of quality in testing practices.

Standardization Process

1. **Template Creation:** Develop a standardized template for creating STPs. This template should include sections for project overview, testing objectives, testing strategy, test environment details, roles and responsibilities, and other pertinent information.
2. **Documentation Standards:** Establish clear documentation standards to ensure consistency in how information is presented across different sections of the STP. This includes standardized formats for test cases, test scripts, and test reports.
3. **Review Processes:** Implement review processes to ensure that STPs adhere to the established standards. Regular reviews by experienced testers or quality assurance professionals can identify deviations and provide valuable feedback for improvement.
4. **Training Programs:** Conduct training programs to familiarize testing teams with the standardized STP template and documentation standards. This ensures that all team members follow the same conventions and contribute to a unified testing approach.

Benefits of Standardization

1. **Consistency:** Standardization promotes consistency in testing processes and documentation, reducing the likelihood of errors and misunderstandings.
2. **Efficiency:** A standardized STP accelerates the testing process by providing a structured and familiar framework. This efficiency is particularly valuable in fast-paced development environments.
3. **Collaboration:** Teams working on different aspects of a project can seamlessly collaborate when using standardized STPs. This enhances communication and ensures that everyone is on the same page regarding testing objectives and strategies.
4. **Quality Assurance:** Standardization contributes to overall quality assurance by instilling best practices and proven methodologies into the testing process. This, in turn, improves the reliability of testing outcomes.

Usage of Software Test Plans

The usage of Software Test Plans extends beyond the testing team. It serves as a comprehensive reference and communication tool for various stakeholders involved in the project.

Stakeholders

- **Users:** In the context of the Capstone Management System, users encompass students, faculty, mentors (representatives), and administrators. For these stakeholders, the STP acts as an assurance of the platform's reliability, functionality, and security. It assures students that their project details are secure, faculty and mentors that the system facilitates effective project management, and administrators that the system aligns with the department's goals.
- **Investors:** The department, acting as investors, relies on the STP to understand how the testing process aligns with the project's overall goals. The STP provides insights into the strategies in place to ensure the software meets the department's investment expectations.

- **Maintainers:** Developers, as maintainers, find the STP invaluable in understanding the testing approach, expectations, and potential areas of focus for development. The STP guides developers in creating software that aligns with the intended testing scenarios.

8.7 Acknowledgments

This collaborative Software Quality Assurance Plan has benefited from the collective efforts of the development team, testing team, and all stakeholders involved in ensuring the success of the Capstone Management System.

8.8 Conclusion

In the ever-evolving domain of software development, the meticulous crafting and strategic deployment of a sophisticated Software Test Plan (STP) alongside a nuanced testing methodology are integral to project success. This strategic approach involves precision in defining the release scope, adhering to well-planned timelines, outlining specific test objectives, and delivering comprehensive testing artifacts. A well-designed test strategy and careful planning of the testing environment and data contribute to the development of a robust and dependable end product.

The standardization of STPs enhances operational efficiency and fosters collaborative excellence by instilling consistency and quality assurance throughout the testing lifecycle. Stakeholders, including users, investors, and maintainers, rely on the STP for critical insights, elevating its role beyond a mere testing document to a vital tool for project communication and alignment.

In the expansive landscape of software testing, a rich array of methodologies assumes a pivotal role in validating the functionality, security, and usability of software products. Ranging from functionality and usability assessments to performance and security evaluations, the testing toolkit offers a comprehensive array of quality assurance mechanisms. Navigating through diverse testing types such as unit testing, integration testing, and system testing, the overarching objective remains unwavering – the delivery of a software product that not only meets but exceeds user expectations.

Through a judicious blend of meticulous planning, flawless execution, and adaptable testing strategies, development teams navigate the intricate terrain of software testing, fostering innovation, reliability, and, ultimately, user satisfaction. The symbiotic relationship between a meticulously designed STP and a diverse testing approach forms an unassailable framework, guiding development teams toward the triumphant delivery of software solutions in today's dynamic and demanding technological landscape.

CHAPTER

NINE

TESTING STRATEGY FOR THE CAPSTONE MANAGEMENT SYSTEM (CMS)

To ensure the reliability, security, and user-friendliness of the CMS, a comprehensive testing strategy is essential. The strategy encompasses several levels and types of testing, each targeting specific aspects of the system.

our levels of testing include:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

9.1 Unit Testing:

- Purpose: Unit Testing serves as the first level of testing and is critical for ensuring that each individual component, or “unit,” of the Capstone Management System (CMS) functions correctly. The objective is to detect problems early in the development process, which can reduce the cost and complexity of fixing bugs later.
- Approach: Developers are responsible for writing unit tests, which are small and automated tests that cover individual functions within the CMS codebase. These tests are created concurrently with the development of system features. A unit test might involve testing a single function with various inputs and validating the outputs against expected results. For instance, if a function is designed to register a new user, the unit test would check to ensure that with valid input data, the user is correctly added to the system, and with invalid data, the proper error is returned. Unit tests are automated, allowing them to be run frequently throughout the development process. They are ideally idempotent, meaning they can be run any number of times without affecting the system state.
- Involvement: The development team is the primary agent of unit testing. This team includes the individual developers who write the tests for their own code, ensuring that each unit is tested before being integrated with the rest of the system. This practice adheres to the fundamental principles of test-driven development (TDD) and agile methodologies.

9.2 Integration Testing :

- Purpose: Integration Testing in the CMS is essential for verifying that different units or modules of the application work together as intended. It is designed to expose any faults in the interaction between integrated units after they have been individually unit tested.
- Approach: Integration tests are more complex than unit tests and require a comprehensive understanding of the system's architecture. These tests focus on the points of interaction between different modules, such as database access layers, APIs, and user interface components. For example, after unit testing individual APIs, an integration test would ensure that the API layer communicates correctly with the backend database and that the front-end user interface correctly displays the data retrieved from the API.
- Involvement: This level of testing is typically conducted by both developers and QA engineers. Developers may create integration tests that are run in a developer's environment, while QA engineers design more elaborate test scenarios that simulate real-world usage in a staging environment, bridging the gap between unit testing and system testing.

9.3 System Testing

- Purpose: The purpose of System Testing is to validate the CMS in its entirety against the specified software requirements. This is the first level of testing where the application is tested as a complete system.
- Approach: System testing is conducted in an environment that closely resembles the production environment where the CMS will ultimately be deployed. This includes the actual hardware, software, network configurations, and other system components. The test cases for system testing are derived from the system's requirements and are designed to cover all the functional and non-functional aspects of the CMS. For instance, system testing will verify the end-to-end workflows for project proposal submissions, user authentication, project tracking, and reporting features.
- Involvement: The QA team, often with support from systems engineers, typically takes the lead on system testing. Their independent assessment is crucial to ensure objectivity and that the system meets the quality standards required for delivery. They work in close cooperation with project stakeholders to ensure that all requirements are addressed.

9.4 Acceptance Testing:

- Purpose: Acceptance Testing is the final phase of testing before the system is released to the users. It aims to ensure that the CMS meets the business requirements and is ready for operational use.
- Approach: Acceptance testing is user-centric and focuses on the usability, functionality, and integrity of the CMS. It is performed in an environment that mirrors the production setting and involves testing with real-world scenarios to validate the system against user requirements. For example, acceptance testing may involve faculty and students performing end-to-end tasks such as submitting and reviewing capstone projects to confirm that the CMS behaves as expected.
- Involvement: End-users, stakeholders, and sometimes a select group of actual customers are involved in acceptance testing. Their feedback is crucial for the QA team to fine-tune the system before the final release. This collaborative approach helps ensure that the CMS is not only technically sound but also aligns with the user's expectations and preferences.

CHAPTER

TEN

DETAILED TESTING METHODOLOGIES FOR THE CAPSTONE MANAGEMENT SYSTEM

In the pursuit of excellence for the Capstone Management System (CMS), a multifaceted testing strategy is implemented. This strategy encompasses various types of testing to ensure the CMS not only meets its functional requirements but also delivers a superior user experience, performs reliably under stress, and maintains the highest standards of security.

10.1 Types of Testing:

- Functional Testing:

This type of testing is centered on verifying the CMS's functionalities against defined specifications. Each function the system is supposed to perform is tested by providing appropriate input to determine if the output is as expected.

Execution: Functional tests are conducted both manually, to catch unforeseen issues, and automatically, to ensure consistent test coverage across all features. For instance, automated scripts may be used to test user registration, project submission, and administrative controls, while manual testing may focus on more complex scenarios that require human judgment.

- Usability Testing:

Usability testing evaluates the CMS's interface and overall user experience. It aims to confirm that the system is intuitive, user-friendly, and accessible to all users, including those with disabilities.

Execution: Actual user sessions are conducted to observe how users interact with the CMS. These sessions help identify areas where users encounter difficulties or confusion, guiding improvements to the user interface and workflow. Techniques such as A/B testing, user interviews, and heuristic evaluations may be employed.

- Performance Testing:

Performance testing assesses the CMS's responsiveness and stability under various conditions. It is crucial for ensuring that the system operates quickly and reliably, even when handling large numbers of concurrent users or processing sizable datasets.

Execution: Automated tools simulate high-traffic conditions, stress the system beyond its normal operational capacity, and measure how well it scales as the load increases. Performance metrics such as response time, throughput, and resource utilization are analyzed to identify performance bottlenecks.

- Security Testing:

Security testing is vital to identify any vulnerabilities within the CMS and to ensure that data protection mechanisms are effective.

Execution:

Mitigation of Attack Surfaces: Regularly scanning the CMS to identify and reduce the potential entry points for unauthorized access. Functional Testing of Privacy: Verifying that privacy controls, such as RBAC and data encryption, are properly implemented and cannot be bypassed. Database Security: Employing tests to protect against threats like SQL injection and to ensure that sensitive data is not exposed through the CMS.

10.2 Manual and Automated Testing in the Capstone Management System

- Manual Testing:

Manual testing is a hands-on process whereby testers manually execute test cases without the use of automated tools. This approach is particularly favored in the CMS project for:

In the early stages of development, when the behavior of new features is still being defined, manual exploratory testing is invaluable. Testers use their creativity and intuition to experiment with the system, identifying defects that are not covered by test cases.

Manual testing is essential for assessing the CMS's user interface and user experience. It involves real user interaction, providing insights into the system's navigability, design, and overall user satisfaction.

Certain tests, due to their intricate nature or because they require a high level of human judgment, are conducted manually. These may include complex integration scenarios or tests that involve hardware interfaces.

Manual testing is utilized in the CMS for levels of testing where human insights are crucial and where automated testing may not be feasible or cost-effective. It allows for a nuanced assessment of the CMS, ensuring that the system is not only technically competent but also aligns with user expectations and provides a quality experience.

- Automated Testing

Automated testing involves the use of specialized software to execute test cases and compare actual outcomes with predicted results. The CMS employs automated testing primarily for:

As new features are added to the CMS, automated regression testing ensures that new changes do not break or degrade existing functionality. This is crucial for maintaining system stability over time.

Automated testing tools are used to simulate multiple users or high-load scenarios, assessing the CMS's performance, such as its response times and throughput under various conditions.

For tests that need to be executed repeatedly over the course of the CMS's development lifecycle, automation ensures these tests are performed consistently and efficiently.

Automated testing in the CMS is justified by its ability to perform a large number of tests rapidly and with high precision, which is especially important for regression and performance testing.