**Documentation: Implementing SCM for a Student Group Website.**

**Developers:**

Jerry Ochieng: BSE-0005/2022

Elvis Mureithi: BSE-0046/2023

Victor Murithi: BSE-0043/2023

**1. Identifying Configuration Elements**

a. The files that provide a website's structure and functionality are called the website codebase. This would include PHP files, theme files, plugins, JavaScript, CSS, and any other relevant files for a WordPress website.

b. Content Files: These files include the text, photos, videos, and other types of material seen on the website. On a WordPress website, this might consist of blog entries, page content, media uploads, etc.

c. Database: The website uses the database to store structured data such as user profiles, preferences, content, and other dynamic data. This is usually a MySQL or MariaDB database used by WordPress to hold user accounts, configuration settings, pages, posts, and comments.

d. Documentation: This covers any material about the website, including development notes, setup instructions, configuration manuals, and documentation for themes and plugins. Accurate documentation is crucial for maintaining and comprehending the website's configuration and operation.

**2. Configuring a repository for Git:**

a. Selecting a Git Hosting Provider: To host the repository, choose a Git hosting provider like Bitbucket, GitHub, or GitLab. Because of its versatility and simplicity, university student organisations often select GitHub.

b. Set up the Git repository: Go to the website's root directory to gain access to the server that hosts it. Using the 'git init' command, create a new Git repository from scratch.

c. Incorporate Configuration Items into the Archive:

- Use 'git add' to add the website's codebase files to the repository.

Similarly, provide content files with pictures, movies, etc.

- Export the first set of data and the database schema to SQL files, then add them to the repository.

Please attach any pertinent documentation files.

d. Commit modifications: Use 'git commit -m "Initial commit"' to commit the changes after adding the required files to the repository.

e. Create a Remote Repository: On the selected Git hosting service, link the local Git repository to a remote repository. Usually, the command 'git remote add origin <repository\_URL>' is used for this.

f. Push Changes to Remote Repository: After everything is finished, use 'git push -u origin master' to push the committed changes to the remote repository (assuming the main branch is called "master").

g. Configure Branches and Permissions: Depending on the size and complexity of the organization, you may set up branch protection rules and permissions to manage who may merge changes into certain branches.

**3. Creating Models of Branching:**

a. Development Branch: Developers combine their feature branches into the development branch, the primary integration branch. It might have some ongoing development, but it should be mostly stable. From the development branch, developers generate feature branches.

b. Feature Branches: A separate feature branch should be created for every new feature or bug patch that branches off from the development branch. This allows developers to work individually without affecting the modifications made by others. Before a feature is merged into the development branch, it must pass code review and testing.

c. Staging Branch: Changes are tested in a setting similar to production using the staging branch. When they are considered ready for release, they are merged into the staging branch for final testing and approval.

d. Production Branch: The production branch represents the live website. Changes from the staging branch are only merged into the production branch once they have been extensively tested and authorized. This ensures that the live environment only receives stable and confirmed updates.

**4. Putting in Place Formal Processes for Code Review, Pull Requests, and Merge to Production:**

a. Pull Requests (PRs):

Developers submit pull requests from their feature branches to the development branch. PRs should clearly describe the modifications performed and any relevant background information.

- Pull requests start automated tests and inspections to ensure the suggested modifications meet quality requirements.

b. Code Review: Before being merged, pull requests are reviewed by other team members. Code review facilitates knowledge sharing among team members, ensures code quality, and finds flaws.

- Reviewers provide comments and ideas for enhancements, and conversations could occur in the pull request thread.

c. Testing: Pull requests should trigger automated tests as part of the continuous integration (CI) process. Unit tests, integration tests, and other pertinent tests fall under this category. Manual testing should be performed in addition to automated testing, particularly for updates that impact the user interface or vital functionality.

d. Approval Process: Before changes are merged into the development branch, they must be approved by one or more team members. The modifications are approved when they satisfy the team's compatibility, functionality, and code quality requirements.

- For modifications to be approved and merged into the staging branch, they may need to undergo extra testing, including stakeholders or QA testers.

e. Merge to Production: Modifications may only be merged into the production branch after they have been carefully examined, validated, and tested in the staging environment. To guarantee consistency and dependability, the merging to production should be carried out by a dedicated team member or via an automated deployment procedure.

**5. Establishing Baselines at Project Milestones and Major Releases:**

a. Essential Releases: Major releases, which are characterized by the addition of new features, notable enhancements, or modifications to the general design, usually indicate crucial turning points in the project's development. At significant releases, to establish baselines:

- Establish a release schedule specifying significant releases' dates and contents. This timetable might be based on a feature-based strategy (releasing once a set of planned features is completed) or a time-based approach (quarterly releases, for example).

-To stabilize the codebase, freeze the development branch before each major release. This guarantees that only essential problem fixes and modifications are made prior to release.

- To designate the codebase as a baseline for the major release, tag it in the version control system (like Git) with a unique identifier (such as a version number or release name).

- For future reference, list the enhancements, new features, and modifications made to the release.

b. Project Milestones: These are critical junctures in the project schedule, including finishing a particular feature, entering a development phase, or accomplishing a predetermined objective. At project milestones, establish baselines:

Establish benchmarks in accordance with the project's goals and development schedule. These benchmarks must be quantifiable and show the project's progress toward completion.

Assessing the project's current status will ensure that the codebase is stable and satisfies the milestone goals at every milestone.

- To designate the codebase as the milestone's baseline, tag it in the version control system.

- Record the accomplishments, difficulties faced, and lessons discovered during the milestone to benefit future reference and project retrospectives.

**6. Offering Dashboards and Tools for Change Monitoring and Release Control:**

a. VCS, or version control system:

Use a version control system (VCS) like Git, which has built-in branching, versioning, tagging, and change-tracking features.

- Utilize branching and tagging techniques, such as feature branches, release branches, and version tags, to arrange and monitor codebase changes across time.

b. Issue Tracking System: Organize and monitor tasks, defects, and feature requests using an issue tracking system (such as Jira, Trello, or GitHub Issues).

- Establish traceability between code changes and project requirements by linking problems to releases and code changes.

c. Pipelines for Continuous Integration and Continuous Deployment (CI/CD):

- Construct CI/CD pipelines to automate deployment, testing, and build procedures.

- Automate tests, conduct code quality checks, and release changes to staging and production environments using continuous integration/continuous delivery (CI/CD) solutions like Travis CI, GitHub Actions, or Jenkins.

d. Release Management Tools: Use release management tools to schedule, monitor, and record releases (such as GitHub Releases and GitLab Releases).

- Make release notes or changelogs that list each version's enhancements, new features, and problem fixes.

e. Dashboard and Reporting Tools: - To see project progress, including task status, code modifications, and releases, create custom dashboards or utilize project management tools.

- Give stakeholders access to dashboards and reports so they can monitor project progress, detect problems or bottlenecks, and monitor milestones.

**7. Automating New Release Builds and Deployments:**

a. Continuous Integration(CI):

- Create a continuous integration (CI) server (such as Jenkins, GitLab CI, or GitHub Actions) to automate the build process sparked by codebase changes.

- Set up continuous integration (CI) pipelines to get the most recent code from the repository, build it, run automated tests, and produce artefacts (such as packaged apps and generated binaries).

b. Continuous Deployment (CD): Implement CD pipelines to automate the deployment of new releases to staging and production environments.

- Create deployment scripts or use deployment automation technologies (such as Chef, Ansible, and Puppet) to automate server configuration and application update deployment.

c. Deployment techniques: - To reduce downtime and risk during releases, use canary or blue-green deployment techniques.

- Blue-green deployment entails moving traffic from the previous environment (blue) to the newly deployed program version in a different environment (green).

Before completely deploying to all users or servers, canary deployment involves progressively rolling out the new version to a selection of users or servers and monitoring for any difficulties.

d. Automated Testing: - Before deploying the application, include automated tests in the CI/CD pipelines to verify its functionality and quality.

- Unit tests, integration tests, end-to-end tests, and performance tests are examples of automated testing.

e. Monitoring and Rollback: - Include automatic rollback procedures to return to the prior version in case of deployment failures or unforeseen difficulties. - Integrate monitoring and alerting technologies (e.g., Prometheus, Grafana, ELK stack) into the CI/CD pipelines to identify issues during deployment.

**8. Creating Reports for Auditing and Status Monitoring:**

a. Tools for Dashboard and Visualization:

Use dashboard and visualization tools to create reports for auditing and progress tracking.

- Create dashboards that provide important information, such as system health indicators, build status, test results, and deployment status.

b. Custom Reporting: - Create reports specifically suited to the project's and its stakeholders' requirements.

- Release activity summaries, deployment histories, test coverage, and performance metrics are examples of possible report contents.

c. Integration with Project Management Tools: To provide a thorough picture of project status and advancement, integrate reporting tools with project management platforms (such as Jira and Trello).

- Automatically produce reports using information from CI/CD pipelines, version control systems, and project management tools.

d. Audit Trails: - Create audit trails to monitor system settings, deployments, and codebase modifications.

- Keep logs of every activity during the development, testing, and deployment stages for auditing reasons.

e. Scheduled Reports and Alerts: - Establish scheduled reports and alerts to inform relevant parties of noteworthy occurrences, such as deployable failures, successful releases, or severe system problems.

- Reports and alerts may be sent via chat notifications, email, or integration with the team's collaborative platforms.