Clubby
Configuration and Change Management Report

1 Introduction

Configuration management is concerned with the policies, processes, and tools for managing changing software systems. We need to manage evolving systems because it is easy to lose track of what changes and component versions have been incorporated into each system version. Versions implement proposals for change, corrections of faults, and adaptations for different hardware and operating systems. If you do not have effective configuration management procedures in place, you may waste effort modifying the wrong version of a system, deliver the wrong version of a system to customers, or forget where the software source code for a particular version of the system or component is stored. Configuration management is needed in this project due to the reasons listed. The change management process is initiated when a customer completes and submits a change request describing the change required to the system. This could be a bug report, where the symptoms of the bug are described, or a request for additional functionality to be added to the system. The reason this report is needed is that, during the development process, we need to monitor many different versions of each software component in a configuration management system. In this way, unexpected dangers are prevented. In this report, the purpose of which configuration management is used, how changes are managed and how the deployment process takes place in this project are included. It explains how the configuration management of this project carries out its activities, how it is followed in carrying out these activities, and which technologies are used.

2 Purpose

The development process implements various version control methods and deployment configuration methods. For independent development of each use cases, we implement a plugin/package-based system for different modules.

Configuration management is essential for team projects where several developers are working at the same time on a software system. With the configuration management, team members are entered into the system and the changes made on the code are managed.

In this project, all team members develop own components and codes within the tasks assigned to them. Requests to be changed in the software are followed up with change management. With this follow-up, it is determined when, in which part of the project and how the changes will be implemented. At the same time, it is ensured that the changes made by different developers with the version management do not affect each other. A plugin/package based for different modules is implemented so that members can development each use cases independently. In this way, it is avoided that the members of the team avoid each other from working. Whenever an update or change is made, the distribution of tasks for each team member is made and which components each team member will develop is determined. In this way, conflicts are avoided. When members develop components, the program components are brought together, and an executable system is created, and certain tests are performed to ensure that they are working correctly. If an error is encountered, the component causing the error is reviewed and the error corrected.

Thanks to this way of working, each member is aware of all changes. and updates. In this way, each member knows the final form of the project. It is easier and faster to develop the project this way. Each system and change is logged. If a serious problem occurs or the change is desired to be reversed, old versions of the code can be used.

3 Configuration and Change Management Specifications

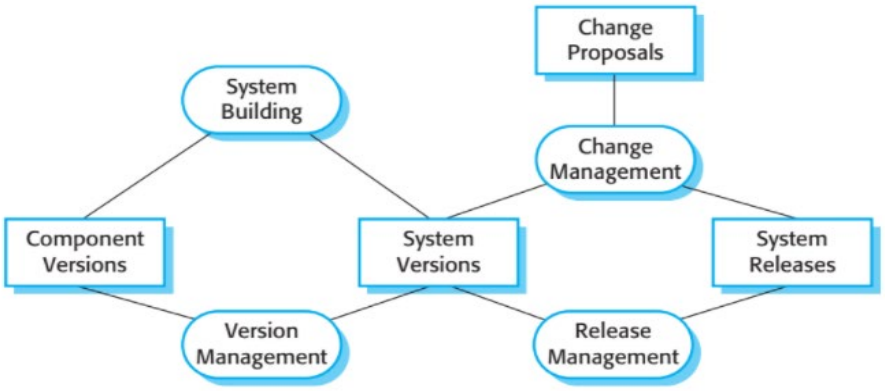


Figure 1: Configuration management activities

Version Control: We implemented version control for collaborative development for both frontend and backend. The main specification is that each artifact generated by developers should fit to system, if not, they are reviewed and updated before merging. Also, for backwards compatibility and new integrations, the system should use package management for third party libraries and various version controlling systems for database, system source code and integration. Each developer pushes code to their branches named as "{developer name}-{component name}" and can request merge. Some of these changes may include systemwide changes (package changes, database changes), these changes should be pointed out in the requests for ease of review.

Change Management: The development process should implement plugin/package-based development for artifacts. By implementing this method, system becomes modular and incase of incompatibility, modules can be extracted from change. The process make use of user stories for sprints and these stories are shared between developers to meet changes at time and developers are informed better by this method. At the end of sprints, each change is subjected to unit tests for testing incompatibility. These tests can be atomic on merging or systemwide on integration. To avoid conflict between development, each developer develops different components however these components may be in same package. The merges are tested to avoid conflict and incompatibility. The review is done by software tester to ensure that change is applicable. Configuration manager reviews milestone updates to ensure that these updates are compatible with old software. The code review is done by software architect for code quality matches with architecture design.

Configuration Management: As system is deployed and tested in different environments, we implemented various configuration settings through environment variables. These settings change the drivers for database, email provider etc. Also, for flexibility, the system should integrate drivers for abstraction of various implementation layers that include job scheduling, database connections etc. For every developer, in collaborative way, each development takes place in a parallel process and to ensure that any conflict is avoided, each developer works in their own components enrolled in meeting before sprints.

Release Management: Each release is tested for integration since new changes may generate new bugs in either old integrations or even worse, unknown bugs. Each update to system is staged before deployment in a staging server, integration is tested in this environment. For quality, configuration manager ensures that the product is compatible to current solution. Software analyst ensures that system requirements and software quality for given update is met. Software tester ensures that system handles as expected in this process. After the review of staged version is done, the new software deployed to deployment environment.

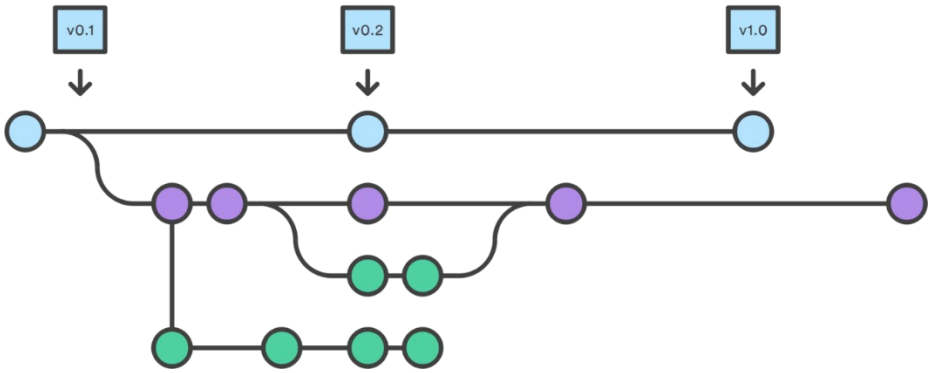
4 Key Considerations

GitHub VCS should be used in a way that each component is developed and updated in its branch. Also, though issues panel, each developer should be informed about broken features, bugs, and incompatibilities. The development process and changes are tracked via the system. Unit tests are utilized to ensure that any change in system is tested for quality and implementations. Also, integration tests should be used for software quality assessment of product.

Database migration system is implemented to track and update current database without rebuilding it for changes. The flyway package is used for this purpose in the system. Also, flyway ensures that migration files are not changed via checksum tracking. On the other hand, to ensure that system uses latest versions of package managers are used. Maven is used for backend and NPM is used for frontend packages. Laravel-mix is used to build frontend code into resource code of backend.

Staging environment is used to build production system and test integration with current system. In case of unexpected failures, end system is not affected from change and change can be delayed. After the staging, product is deployed to the production server.

Environment variables are utilized to change system drivers of development/local and production environments. This ensures that system is flexible for any solution and can be deployed a cloud or a dedicated server.



5 Changes Managed

- Instead of using a third-party calendar application, namely Google Calendar API, as initially planned. It turned out be better and easier to develop a homemade calendar program than integrating an overly complex outside API program into our website. As a result, our plans and some minor designed models got short-reaching changes. Previous integration configurations are deleted, and numerous additional versions have been built on top of them.
- Changed the meeting software from Google Calendar to Zoom and made the necessary changes and configurations to our design and plans to make sure their transitions will be smooth and controlled.
- Data structures we used changed, sometimes incrementally, sometimes at break-neck speeds during development to accommodate unforeseen needs and dependencies, and also to include additional functionalities not envisioned at the planning stages. And to accommodate those data structure changes, the changes to and migrations of databases are also realized.
- A slight change to our Git version control standards was also commenced. Initially, it was decided to be better to use “$git rebase” command in order to combine branches. But over time, thanks to its complexity and danger posed onto our development and productivity that we faced multiple times, it was decided to use “$git merge” command instead as a way of combining branches.

6 References

1. *Change Management*. GitLab. (n.d.). https://about.gitlab.com/handbook/engineering/infrastructure/change-management/
2. *Environment Variables*. Environment Variables (The Java™ Tutorials > Essential Classes > The Platform Environment). (n.d.). https://docs.oracle.com/javase/tutorial/essential/environment/env.html
3. Flyway. (n.d.). Flyway Documentation. https://flywaydb.org/documentation/
4. Sommerville, I. (2018). *Software Engineering*. Pearson.
5. *Spring Data JPA*. Spring. (n.d.). https://spring.io/projects/spring-data-jpa