Configuration Management Plan

1. Introduction

1.1 Identification

The SCM plan is for the PolyChord application's development, and the related testing, version control, continuous integration, and deployment software. For testing, Mocha, Chia, and Karma are used. For version control, Git and Github are used. For Continuous integration, Drone CI is used. For deployment, Docker and Docker Hub are used.

1.2 Purpose

The Configuration Management Plan lays out in detail how the DAJ team will manage the control of configuration items being developed in each phase. It defines both the policies and procedures for configuration management, and the infrastructure necessary to implement them throughout the phases of the project.

1.3 Scope

This version of the Configuration Management Plan is applicable to the Initial Design and Development phases of the PolyChord project. It may be modified for the following phases depending on the Configuration Management requirements for such phases. All PolyChord team members, while working on the PolyChord Project, will adhere to the approach outlined in this project.

1.4 Limitations and Assumptions

The major limitation in this project is having the person-hours to complete all tasks, and finding the time to perform the required meetings, as the members of DAJ are all part-time, and have other jobs. This means that configuration management tasks must be completed as efficiently as possible, and all schedules and timed tasks must be made flexible to accommodate the team's constantly changing schedule. The other limitation is that the team has access to limited resources, so most paid tools and libraries are not an option. For this reason, the team must rely on free and open source projects for most of their support software and development tooling.

2. Management

2.1 Organization

Outline of the organizational context (technical and managerial) within which the configuration management activities are implemented.

2.1.1 Configuration Manager

There will be one member assigned to heading the management of the configuration control for Polychord. Their responsibilities will include: - Heading the maitenence of the Polychord documentation - Approving or denying significant changes to the intended course of the program - Allocating work on revisions of documents when needed Appointment of the Product leader will be based on their understanding of the underlying goals of the application and tenure.

2.1.2 Product Leader

There will be a single member of the development team whose responsiblities include: - Maintaining the overall direction of feature implementation - Referencing the relevant documents to projected features - Allocating work to the development team - Approving or denying significant changes to the source code Appointment of the Product leader will be based on their understanding of the underlying application and tenure.

2.2 Responsibilities/Configuration Control Board

The leaders in section 2.1 shall head relevant control boards, which whill simply be composed of any developer or leader of Polychord who cares to give their input on the direction that the leader is taking the project.

2.2.1 Configuration Management Board

The Configuration Management Board shall be headed by the Configuration Manager, and the goal is much the same as outlined in section 2.1.1. - When changes are in question, they must be approved by either 50% of participating members including the Configuration Manager or by ⅔ of the participating members if not approved by the Configuration Manager. - A meeting of the board is to be called whenever major changes to the direction of Polychord is in question, including removal or addition of features, and changes to general guidelines and regulations.

2.2.2 Product Management Board

The Product Management Board shall be headed by the Product Leader, and the goal is much the same as outlined in section 2.1.2. - When changes are in question, they must be approved by either 50% of participating members including the Product Leader or by ⅔ of the participating members if not approved by the Product Leader. - A meeting of the board is to be called whenever major changes to the codebase of Polychord is in question, including removal or addition of features.

2.3 Applicable policies, directives and procedures

There are no relevant policies, directives or procedures pertaining to Polychord on the management of the Product Board.

3. Activities

3.1 Configuration identification

- Identify configuration items (events, items, procedures)
- Name configuration items (unique identifiers)
- Acquiring configuration items (physical procedures) - Not sure what this would be.

P100 PolyChord System

P110 PolyChord Frontend

P111 Tone.js (External Synthesizer Library)

P112 PapParse (External Parsing Library)

P113 Google Sign-on API

P120 PolyChord Backend

P121 Flask (Python Web Framework)

P122 Redis User Database

P200 PolyChord Development System

P210 Testing stack

P211 Karma (Javascript Test Runner)

P212 Mocha (Javascript Testing Framework)

P213 Chai (BDD/TDD Assertion Framework)

P220 Distribution System

P221 Docker Hub (Distribution Image Builder and Distribution Mechanism)

P222 Python Alpine Docker Image (Distribution Base Image)

P223 Gunicorn (Python WSGI HTTP Server)

P230 Version Control & Integration

P231 GitHub (Version Control)

P232 Drone CI (Continuous Integration Pipeline)

P300 Documentation System

P310 Drone CI (Continuous Integration Pipleine)

P311 Bash Scripts

P320 Markdown to PDF conversion

P321 Gotenberg API (Docker image thecodingmachine/gotenberg)

P330 PlantUML to PNG conversion

P331 Kroki API (Docker image yuzutech/kroki)

3.2. Configuration control

Configuration control is taken care of within GitHub issues and pull requests: Change requests, evaluation, and approval or rejection all takes place within the GitHub issue representing the proposed change, and once the change has been implemented, it's implementation is contained within a pull request (or series of pull requests), so that the changes' impact can be evaluated before it is merged.

3.3. Configuration status accounting

While there is currently no automated Configuration status account mechanism that tracks configuration metrics, such a system is planned, and will track the status of the Continuous Integration pipeline, by both performing a health check on it, and collecting logs from it, to enable easier troubleshooting. In order to keep the source code clean and functional, automatic unit tests and integration tests shall be built into the Continuous Integration pipeline, so all source code changes must undergo an automated testing processes before they reach the repository. Finally, all Pull Requests for the Master branch will require manual review by team members who were not involved in forming the Pull Request, so that further issues with the Configuration items can be monitored and prevented.

3.4. Configuration evaluation and reviews

Configuration evaluation happens when the change is being discussed in it's related GitHub issue, and once an implementation is fleshed out, the review of the implementation is performed by the Product Owner on the Configuration change Pull Request. At least one of these steps (Evaluation and Review) occurs during each sprint per configuration change, though in some cases both may occur within the same sprint if the change is small in size.

3.5 Interface control

The majority of configuration items that are effected by changes outside the scope of the plan are the supporting software, which may receive updates and function changes over time. If possible, such updates should be taken care of within the normal software development cycle, though the software release reason should be taken into consideration when determining the urgency of the update: for example, feature updates should generally be considered low priority unless the new feature is needed for further software development, while security patches should be treated as high priority and incorporated in the very next development cycle, or even out of cycle if the security issue is major.

3.6 Release Management and Delivery

Releases are performed via the GitHub “Release” functionality, and typically happen once every two sprints. A release is a tagged commit on the master branch of the git repo, and the release includes both a zip of the source code, as well as a Docker image, which is auto-built when the release is tagged, and released on Docker Hub, which is the primary mechanism for delivery of the software. Docker Hub and Github both keep previous releases, and allow the releases to be tagged with a version number, which allows the releases to be tracked over time.

4. Schedules

- Sequence and coordination of SCM activities.
- Relationship of key SCM activities to project milestones or events, such as:
    - Establishment of configuration baseline
    - Implementation of change control procedures
    - Start and completion dates for a configuration audit
- Schedule either as absolute dates, relative to SCM or project milestones or as sequence of events.
- Graphical representations can be used here.

4.1 - Coordination

4.2 - Milestones

4.3 -

5. Resources

- Identifies environment, infrastructure, software tools, techniques, equipment, personnel, and training.
- Key factors for infrastructure:
    - Functionality, performance, safety, security, availability, space requirements, equipment, costs, and time constraints.

Environment: Flask (web framework); Redis (database)

Personnel: Devin Christianson, Dawsin Blanchard, Jeremy Thiboutot, Alexander Revello, Alex Feren, Devin Merrow

- Identify which tools are used in which activity

6. Plan Maintenance

The Configuration Manager is responsible for monitoring the plan, and updates to the plan should be performed as necessary, at a minimum once for every major Software release. Changes to the plan can be evaluated and approved through the same GitHub Issue and Pull Request mechanism as is used for Configuration Items, though these artifacts should be tagged as SCM, so that a history of the changes made to the plan can more easily be tracked.