Università degli Studi del Sannio

Sound Recorder
SCM Plan

Assunta De Caro, Pietro Vitagliano
AA. 2020/2021

Table of Contents

1. INTRODUCTION

This document describes the software configuration management activities to be performed in support of Sound Recorder application project.

1.1 PURPOSE

The primary focus of the Software Configuration Management (SCM) is to identify and control major software changes, ensure that change is being properly implemented, and report changes to any other personnel who may have an interest. The objective of SCM is to limit the impact changes may have on the entire system. This will help to eliminate unnecessary changes, and to monitor and control the necessary ones. This allows software development to continue, despite large and/or insignificant changes, lessening development time and resulting in a higher-quality product. The SCM team will oversee these activities, and any changes to existing code or architectural design must pass their inspection before they are carried out.

SCM activities are developed to:

- Identify change
- Control change
- Ensure that change is being properly implemented

1.2 SCOPE

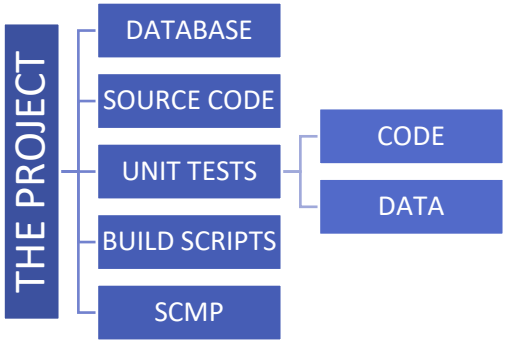
1.2.1 Project Description

The aim of the project concerns the extension and the improvement of the open source Android application Sound Recording, whose rights belong to Kim Daniel and it is distributed under the GPL V.3 license. The product already allows recording, sharing and store management of audio files.

1.2.2 Configuration Item Identification

All the configuration items are related to a single application program and not meant to be automatically generated by a build script, such as UML design diagrams and executables.

CONFIGURATION ITEM HIERARCHY



2. MANAGEMENT

2.1 ORGANIZATION

Role	*Member*
Configuration Manager	Vitagliano Pietro
Change Control Board	De Caro Assunta
Developers	Vitagliano Pietro, De Caro Assunta
Auditor	Vitagliano Pietro, De Caro Assunta

Since we have just two members in our team, we assigned key roles as described above. Both of us is developer, so it's reasonable that each is the other's auditor.

2.2 RESPONSABILITIES

Configuration Manager: This role is responsible for identifying configuration items and CM aggregates and for defining the procedures for creating promotions and releases.

Change Control Board: This role is responsible for approving or rejecting change requests based on project goals.

Developer: This role creates promotions triggered by change requests or the normal activities of development. The main configuration management activity of developers is to check in changes and resolve merge conflicts.

Auditor: This role is responsible for selecting and evaluating promotions for release. The auditor is also responsible for ensuring the consistency and completeness of the release.

3. ACTIVITIES

This section describes in detail the identification of configuration items, the change control process, the process for creating releases and for auditing, and the process for status accounting.

3.1 IDENTIFICATION

As we are extending an existing project, we have kept the configuration items defined by its creator, reported in section 1.2.2, adding the SCMP.

3.2 CONFIGURATION CONTROL

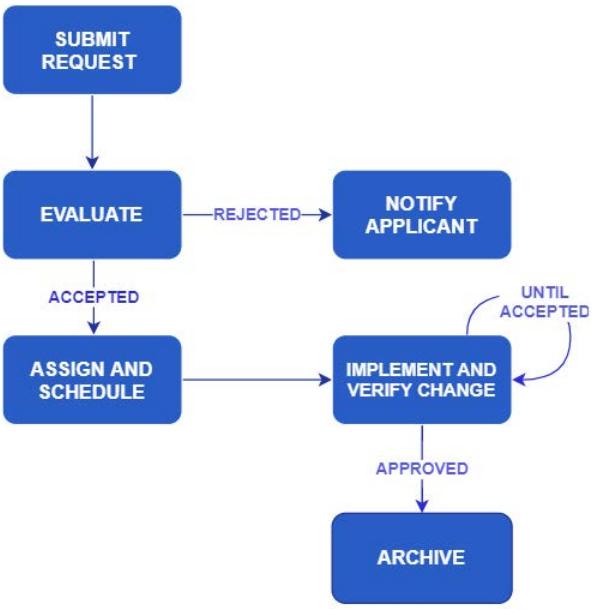
Change request policy: Both team developers and external ones who want to change a baseline, must work within a new parallel branch, and open a *pull request*. An accepted pull request must compile, pass code review, respect coding standards and pass integration testing. Every pull request is assigned to a reviewer. In case of denial, you can discuss the decision with the CCB and resubmit an amended pull request if an agreement is reached.

If you want to suggest a change or open a bug fixing issue, you must at least notify your intentions by official channels. Formal requests will always be accepted, but you don't have to follow a standard form. Every change request includes the applicant's name, the date of the request, the version/component it refers to, the subject of the request, and eventually whether it was approved with its priority (high, medium, low).

Whether for pull request for the issue, the CCB will base his decision on how severely the change will impact the entire system and, more importantly, on the corresponding subsystem. In case of indecision, the CCB can discuss it with the other member. If the CCB deems the change unnecessary, it will contact the person who submitted the request and explain why it was denied.

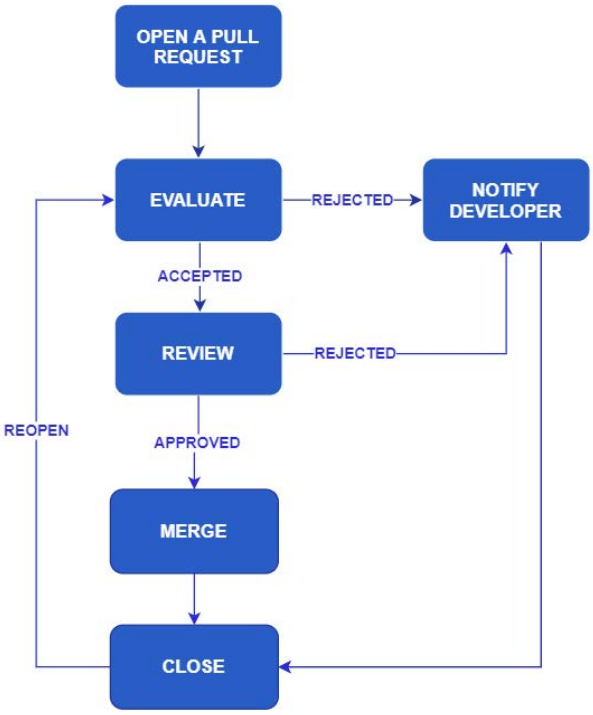
If the issue is accepted, it will be prioritized. assigned to a developer and implemented within a branch parallel to its baseline. If the subject of a request is a bug fix and it is possible to identify the affected portion of the system, then it will be handled ideally by who worked on it.

CHANGE REQUEST FSM



Once implemented, the change request must be filed indicating in addition to the original information, the elements concerned, the verification date and the identification of the new version.

PULL REQUEST FSM



Promotion policy: Developers who want to create a promotion need to work within a branch parallel to a baseline using pull request mechanism. The promotion must contain new features and compile to be considered valid. Optimization is not necessarily required, however critical points within the code must be reported.

Release policy: Promotions to be released must meet the following quality standards:

- They must adhere strictly to code writing guidelines
- They must pass functional testing
- They must be α-tested by both team members.
- They must be β-tested.

Versioning policy: As we extend an existing application, we rely on its versioning system that follows this scheme:

<version>::= <configuration item name><major> <minor><revision>

At the beginning, software is version 1.3.0.

Whenever the system or a subsystem is updated, the program build number (version number) will be updated to reflect the change. The version number will follow a standard x.x.x input mask, with each digit place corresponding to an increasing severity of change.

The third place (x.x.x) will reflect very minor changes to the software. They can be a singular error being corrected, or a minor design change that has little or no impact on the surrounding subsystem.

The second place (x.x.x) will reflect more substantial software changes. They are correction of many minor bugs simultaneously, or design changes that are substantial enough to affect the way the software operates internally or slightly alter the way the user interacts with the interface.

The first place (x.x.x) will reflect severe changes to the software. They can be design decisions that will affect a significant portion of the subsystems and the way they interact with each other, a major change in the functionality of the software, or any interface overhauls severe enough to completely alter the way the user interacts with the interface.

Baseline policy: We mark as baseline every release and every promotion that guarantees the following criteria:

- It must compile.
- All the features expected for that baseline have been implemented, completed, test successfully (unit, integration and system test included).
- All critical points must have been resolved.

3.3 CONFIGURATION STATUS ACCOUNTING

Configuration Status Accounting (CSA) is the process of record, store, maintain, and report the status of configuration items during the software lifecycle. All software

and related documentation should be tracked throughout the software life.

Generally, CI version identifies the modification sheet included in the current version. If necessary, CI is tagged to report extended information to the change.

Source code file requires a slightly different kind of considerations on traceability. Indeed, the software configuration management tool highlights changes with different colour.

The records are stored in a remote repository, which contains all the CI versions. Change requests and issues will be made by and stored on GitHub.

3.4 CONFIGURATION AUDITS AND REVIEW

Auditing: Team members will have a daily report on their individual performance. Every change needs to be discussed with other team members before implemented. There will be in-depth comparisons for the evaluation of promotions, releases, baselines or in view of testing phases.

Code Review: The code review will be conducted combining the use of automatic tools with source code static analysis. The latter will be used to manage criticalities (eg Failure of a build due to faults in the code) and in-depth checks on the implementation of the requirements. If none of them reveals defects, the change will be approved, otherwise it will be rejected. Code review tools will be appropriately configured and called from build script.

4. SCHEDULE

Changes, promotions, releases, will follow the timelines defined by the SCRUM agile life cycle; therefore, any change not foreseen by the current sprint will be scheduled in the next one except for essential changes to the execution of the application. New baseline is expected at the end of every Sprint.

5. RESOURCES

This section identifies tools required for the SCM activity implementation:

Version Control System	Git - free and open source distributed version control system Source Tree – Git GUI client
Change Request	GitHub - In addition to managing the Git-based remote repository, it handles and records issues and pull request
Code Review	CheckStyle, PMD, Gerrit - static code analysis tool
General communication	Slack
Continuous Integration	Trevis CI - is a hosted continuous integration service used to build and test software projects hosted at GitHub.

6. PLAN MAINTENANCE

The maintenance manager of the SCMP is the CM. Furthermore, the frequency of updates will depend on changes in promotion, release, baseline definition and versioning policies. Each change will be followed by an update of the document.