Software Configuration Management Plan

for

Cafe Bunny

SCM_PRO_008_V1.1
Version 1.1 approved
Prepared by Team CodeNation

Nanyang Technological University,
School of Computer Science & Engineering
21/03/2021

Submitted to:

Dr. Shen Zhiqi, Lab Supervisor

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Chew Zhi Kang	17/03/2021	Bachhas Nikita	18/03/2021	Software Configuration Management Plan 1st Draft (Sections 1-3.1.1)
1.1	Bachhas Nikita	21/03/2021	Chew Zhi Kang	21/03/2021	Software Configuration Management Plan 1st Draft (Sections 3.1.1-5.4)

TABLE OF CONTENTS

1 Identification

1.1 Document overview

This document holds the Software Configuration Management Plan (SCMP) for Cafe Bunny application.

1.2 Abbreviations

Abbreviations	**Descriptions**
SCMP	Software Configuration Management Plan
PM	Project Manager
LD	Lead Developer
FD	Front-end Developer
BD	Back-end Developer
QAM	Quality Assurance Manager
QAE	Quality Assurance Engineer
RM	Release Manager
SCI	Software Configuration Item
SDLC	Software Development Life Cycle
FBL	Functional Baseline
ABL	Allocated Baseline
PBL	Product Baseline
SRS	Software Requirement Specification
RFC	Request for Change
CSA	Configuration Status Accounting
VDD	Version Description Document

2 Organization

The Cafe Bunny Team manages the software configuration using tools such as MediaWiki, GitHub, and Google Drive. Responsibilities are shared among the following roles:

- Project Manager (PM)
- Lead Developer (LD)
- Front-end Developer (FD)
- Back-end Developer (BD)
- Quality Assurance Manager (QAM)
- Quality Assurance Engineer (QAE)
- Release Manager (RM)

2.1 Activities and responsibilities

The tables below describe the functions required to manage the configuration of the software, as well as the person in-charged and responsible for the various functions.

Activities conducted when setting up the project	**Person responsible**
Name configuration items	QA
Install the bug repository tool and set up the database	LD
Install the software configuration repository tool and set up the database	LD
Manage and structure the reference space	QAM
Define the configuration processes	QAM

Activities conducted during the project lifecycle	**Person responsible**
Export components for modification, test, or delivery	QAM
Set under control validated components	QAE
Create version, write version delivery document	RM
Approve reference configurations	PM
Verify delivery version and authorise deliveries	PM
Backup spaces	LD
Do configuration audits	QAM
Inspect configuration records	QAM
Archive reference version	RM
Fix bugs and errors	FD, BD
Devise test plans	QAE

Management activities	**Person responsible**
Manage versions and archives	RM
Manage configuration records	RM
Produce reports and statistics	RM
Manage reference space and its access control list	RM
Manage spaces backup and archive media	RM
Manage quality reports	QAM

2.1.1 Decisions Process and Accountabilities

Responsibilities during reviews, audits and approvals listed below:

At the end of a project activity:

Activities	Person Responsible
Do a configuration freeze	RM
Present a configuration state of the components affected by the activity	RM
Present a documentation state of the components affected by the activity	QAM

During a configuration management process audit:

Activities	Person Responsible
Do the configuration management process audit	PM
Present the records of the configuration management process	RM
Present the quality records of the configuration management process	QAM
Present the records of the management process for documentation	RM

3 Configuration identification

3.1 Identification rules

3.1.1 Identification rules of configuration items

3.1.1.1 Identification of a configuration item

The identification of software configuration item (SCI) will have the following format: "XXX.vM.N.A"

1. XXX is the configuration item name
2. M is the major version number of the SCI
3. N is the minor version number
4. A is the incremental version number, if necessary

3.1.1.2 Version number of a configuration item

The attribution of a version number is a prerequisite to any delivery of any configuration item. This number must increment before a new delivery if the product or its documentation have changed.

The definition rules of a version number are the following:

1) Number "M" increases when the team makes major changes to the application. Examples include:
    1. Adding new functions
    2. Potentially backwards-incompatible change to a software package
2) Number "N" increases when the team makes minor changes to the application. Examples include:
    1. Adding small features that do not affect the main function of the application
3) Number "A" increases when the team makes a fix to the earlier minor release. Examples include:
    1. Modifying existing functions to fix bugs

3.1.2 Identification rules of SOUPs

3.1.2.1 Identification of a SOUP

SOUP stands for Software of Unknown/Uncertain Pedigree/Provenance and is a software that has not been developed with familiar software development processes or methodology, or has unknown or no safety-related properties.

The identification of a SOUP will have the following format: "XXX.vM.N.A"

1. XXX is the SOUP name

2. M is the major version number of the SOUP
3. N is the minor version number
4. A is the incremental version number, if necessary

3.1.2.2 Version number of a SOUP

The attribution of a version number is a prerequisite to any delivery of any SOUP. This number must increment before a new delivery if the product or its documentation have changed.

The definition rules of a version number are the following:

1) Number "M" increases when the team makes major changes to the application. Examples include:
    1. Adding new functions
    2. Potentially backwards-incompatible change to a software package
2) Number "N" increases when the team makes minor changes to the application. Examples include:
    1. Adding small features that do not affect the main function of the application
3) Number "A" increases when the team makes a fix to the earlier minor release. Examples include:
    1. Modifying existing functions to fix bugs

3.1.3 Identification rules of documents

3.1.3.1 Description of documents identifiers

The identification of documents will have the following format: "XXX_A_B_C"

1. XXX is the document name
2. A is the document type
    a. REQ stands for all documents pertaining to the customer's requirements (Requirements Documentation)
    b. PRO stands for all documents pertaining to the project's development (Project Documentation)
    c. DIA stands for all documents with diagrams and models drawn, such as the Architecture diagram or the Use Case Model (Diagrams Documentation)
    d. TEC stands for all documents that contain something related to the code, relevant algorithms or interface (Technical Documentation)
    e. USER stands for all documents that are user manuals or guidelines for the user on how the software functions (User Documentation)
3. B is the document number, which is an incremental number, with a separate list for each document type.
4. C is the revision index, is the approved iteration of the document. An example of the revision index is V1 for the first iteration, V2 for the second and so on.

3.1.3.2 Definition and evolution of the revision index

The attribution of a revision index is a prerequisite to any delivery of a document or file. The index shall be incremented before the diffusion of a modified document.

The definition rules of a version number are the following:

1) The revision index increments in the fashion of V1, V2 and so on, when the team makes major changes to the documentations. Examples include:
    1. Adding new elements to the document
    2. Revising any elements in a document supporting potentially backwards-incompatible change to a software package
2) The revision index increments in the fashion of V1, V2 and so on, when the team makes minor changes to the documents. Examples include:
    1. Adding or revising small elements in the document that do not affect any major portion of the document

3.1.4 Identification rules of a media

This includes any medium of communication, such as a CDROM or tape.

3.1.4.1 Internal identification

The identification of a media is described below: “XXX_D_E”

1. XXX is the configuration item identification
2. D is the media number
3. E is the volume, which is an incremental number to distinguish the media if the delivery contains more than one media

3.2 Reference configuration identification

Each reference configuration is defined by:

- An identifier
- Its content listed in the corresponding Version Delivery Description document
- The acceptation or validation reviews associated to the building of the reference configuration

A reference configuration is established for each design review and each test review of the project.

3.3 Configuration Baseline Management

A baseline identifies an agreed-to description of the attributes of a system at any given point in time and provides a known configuration to which the changes are addressed to. The project has three different baselines that are established.

The different baselines to be established and when and how they will be defined and controlled are:

1. Functional Baseline (FBL):
   This defines the functional characteristics of the overall system and the verification required to demonstrate the achievement of those specified characteristics. This baseline has to be established during the System Requirement Specification (SRS) document after the Requirement Specification stage. The baseline has to be controlled by ensuring that each feature added to the software meets the required characteristics needed in the baseline and testing of each feature has to be carried out after each iteration of the code to ensure the baseline is met.
2. Allocated Baseline (ABL):
   This defines the design of the functional and interface characteristics for all system elements and the verification required to demonstrate the achievement of the specified design. This includes all functional and interface characteristics that are allocated from the top-level system or higher level configuration items, derived and analysed requirements, interface requirements with other configuration items, design constraints and the verification required to demonstrate the traceability and achievement of specified functional, performance and interface characteristics. This is established in the detailed design of the software and is controlled by ensuring that every feature developed is similar to the detailed design, as well as carrying out tests which are necessary to verify and validate each configuration item's performance.
3. Product Baseline (PBL):
   This defines the completed and accepted system components and documentation that identified these products. This baseline describes all necessary physical characteristics and selected functional characteristics needed from product testing. The baseline is traceable to the System Requirements Specification (SRS) documents and is fully in the project plan, which lists individual components in the project. The baseline is controlled by ensuring the software developed adheres code-to specifications that is established initially. Technical audits done allows for verification of this baseline.

4 Configuration control

To manage configuration changes and variances in configurations, all changes must be proposed to and approved by the project member, before being carried out. All previous versions of the configurations must be stored and recorded securely. Each configuration item or document will have a revision/variance number that allows for easy recognition of the file. The project manager and the quality assurance manager must oversee that all documents, codes and other entities during the software development life cycle are systematically managed, organised and controlled.

4.1 Change Management

This is the process for controlling changes to the baselines and for tracking the implementation of those changes.

Problem Resolution:

1. Team member must identify what the problem is, what changed have to be carried out and all related documentation that must be modified to support the change
2. Any change to be made to the software or the documents have to be reviewed, evaluated and approved by the project manager before being carried out.
3. Change requests are emitted by the project manager according to the problem resolution.
4. When a change request is accepted by the project manager, a branch is created in the SCM.
5. The identification of a branch will have the following format: "XXX.A.B.C", where:
    a. XXX represents the file or document name
    b. A.B.C represents the revision number of the branch. Each branch will add a new ".number" behind that original format.
    c. For example, there is a first branch from the original line of development, it will be named as A1.1. If there is another branch emerging from the first branch that has already occurred, it will be named as A1.1.1. This process will continue on with more numbers being linked to the back of the name as branching occurs.
6. The branch content is a line of development that exists independently of another line, yet still shares a common history and can be merged together in the future with that common line.
7. Record the new branch's name and a description of its content as well as the modification date in a secure document to allow for easy traceability.

Multiple Configuration:

1. Team member must identify what the problem is, what changed have to be carried out and all related documentation that must be modified to support the change
2. Any change to be made to the software or the documents have to be reviewed, evaluated and approved by the project manager before being carried out.
3. Change requests are emitted by the project manager according to the production procedure.
4. When a change request is accepted by the project manager, a branch is created in the SCM.
5. The identification of a branch will have the following format: "XXX.A.B.C", where:
   a. XXX represents the file or document name
   b. A.B.C represents the revision number of the branch. Each branch will add a new ".number" behind that original format.
   c. For example, there is a first branch from the original line of development, it will be named as A1.1. If there is another branch emerging from the first branch that has already occurred, it will be named as A1.1.1. This process will continue on with more numbers being linked to the back of the name as branching occurs.
6. The branch content is a line of development that exists independently of another line, yet still shares a common history and can be merged together in the future with that common line.
7. Record the new branch's name and a description of its content as well as the modification date in a secure document to allow for easy traceability.

4.2 Interface Management

3rd Party interfaces that need to be managed:

1. Google Maps Application Programming Interface (API) on the homepage of the software application

Procedures for identification of interface requirements:

1. Identifying the interfaces
2. Defining the interfaces:
   This includes the characteristics of each system at the interface, the different types of media involved in the interaction between the user and the interface and the different ways to access the interface.
3. Defining what an interface requirement is:
   An interface requirement is a system requirement that involves an interaction with another system.
4. Identifying systems that interact with the system at the interface:
   a. These interfaces have to be written in pairs.
   b. Do not use the word *interface* in these statements.

c. Multiple interactions can occur between two same systems, but each interaction has to be written as a separate interface requirement.
d. Record and store this in the System Requirement Specification document
e. One such example is: the firebase database system interacts with the Google Maps API system when it stores the coordinates for all existing cafes within a specified radius.

4.3 Evolutions control of SOUP items

Evolutions of SOUP can be managed by:

1. Each version of the SOUP has to be fit for the purpose of the project by ensuring that is meets the functional and performance requirements of the project
2. Find the list of dependencies for the SOUP and ensure that the dependencies stays the same throughout the evolution of the SOUP
3. Perform risk analysis of the SOUP and perform mitigation strategies as required.
4. Conduct unit tests you can link to the specification or unit tests of the library itself after each iteration of the SOUP.

5 Configuration support activities

5.1 Configuration Status Accounting

Configuration Status Accounting (CSA) is the process to 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.

5.1.1 Evolutions traceability

The traceability of modification given their types:

1. Document:
   The modification sheet numbers identify the origin of the modification. The modified paragraphs in the document are identified, if possible, by revision marks.
2. Source File:
   The software configuration management tool records, for each source file or group of source files, a comment where it describes the modification carried out.
3. Configuration Item:
   The Version Delivery Description of the article identifies the modification sheet included in the current version.

5.1.2 Setting up Configuration status

The QAM sets up the state of all versions and of each configuration article with:

1. The label
2. The version number
3. The creation date of the VDD

The QAM writes the VDD.

5.1.3 Configuration status diffusion

The RM and the QAM write the VDD.

5.1.4 Configuration status records storage

The records are stored in a configuration folder, which contains:

1. The requests sorted by record number
2. The software documents
3. The VDD’s
4. The configuration stated sorted chronologically

5.2 Configuration audits

Different types of audits are carried out to assess the compliance of the software with the CM plan. These configuration audits ensure that the software conforms to the required functional and physical characteristics and satisfies the baseline needs.

These reviews include:

1. Peer Review Audits:
   Peer review audits allow for the colleagues of the team to examine the software in order to evaluate its technical content and quality. It is a disciplined engineering practice for detecting and correcting defects in software artifacts, and preventing their leakage into field operations. Peer review audits are most effective when they are performed after each iteration of a configuration item, document or file so as to identify problems that can be fixed early in the software development life cycle. These audits promote high quality in the auditing services. These audits also provide the team with insights and recommendations for any soft of improvement and allows for learning of best practices and ideas from other teams and individuals. Peer review audits exist on a wide spectrum of formality from unstructured activities such as "Buddy Checking" to more informal approaches like walkthroughts.
2. Formal Audits:
   These audits are considered to be more formal and more thorough as compared to peer audits. These audits are generally done by the management team and the stakeholders of the project. Auditors here require each software modification to follow correct protocol with all supporting documents along with it. They ensure that the defined processes are being followed and compliance to the configuration control standards is being made. Auditors here also ensure traceability is maintained during the process and that all changes made to a baseline comply with the configuration status report. These audits serve as a validation of completeness and consistency.

Examples of audits include:

1. Baseline Audits:
   These audits examine each system element to ensure that they meet the required specification that is mentioned in the individual baselines. Actions performed against the baseline is identified, along with the individual involved in any action. The change is then evaluated and remodified to meet the baseline requirements and is audited and subjected for approval again.
2. Functional Configuration Audit:
   A Functional Configuration Audit (FCA) examines the functional characteristics of the software and verifies if the software meets requirements specified in the functional baseline. It consists of reviews of the configuration item's test data and analysis to validate the intended function meets the system requirement specification.
3. Software Configuration Audit:
   This audit is an examination of the software product to verify through testing, inspection and demonstration or analysis results that the product has met the requirements specified in the baselines and in the requirement specification document.

5.3 Reviews

Technical reviews consists of a structured encounter in which a group of technical and qualified personnel analyses or improves the quality of the software product and examines it's suitability for its intended use and identifies discrepancies from approved specifications and standards.

Technical reviews generally consist of 6 stages: planning (choosing of team and materials), orientation (presentations of product, process and goals), preparation (checking of the product), review meeting (consolidation of issues), rework (correction of defects) and verification (verification of product/process quality).

The technical reviews are carried out after every iteration of modification to ensure that the changes meet the required specifications in the baseline. Once a branch has been differentiated from the initial development line, a technical review is carried out to ensure that the branch meets system requirements for the software product.

The configuration managers maintain the configuration management policies, objectives, scope and principles. These policies may be applicable for specific system elements or services. These policies also determine how often configuration audits have to take place. The managers periodically revise the configuration management plan to determine if improvements can be made. They also determine the scope and level of detail for every configuration item maintained.

5.4 Configuration management plan maintenance

There is a need maintaining the configuration management plan as the plan demonstrates that the team understands how to control change and is able to track all changes that were made to the product back to the initial requirement elicited and analysed. An effective configuration management plan provides audit trails and technical review reports to allow for traceability of the change.

These are the activities and responsibilities of configuration management during the lifecycle of this project:

Project Manager:

- Since the PM oversees the entirety of the project, he/she is also responsible for periodically updating the configuration management policies, objectives, scope and principles.
- The PM also needs to ensure that all configuration items related the software and infrastructure
- Control changes to configurations

Lead Developer, Front-End Developer and Back-End Developer:

- The developers of the product should be familiar with all the changes made to the system design and be knowledge of the configuration data from the software

Quality Assurance Manager/Engineer:

- The QAM conducts audits and reviews on the software product after each modification to ensure acceptable software quality
- The QAM responsible for ensuring that guidelines and test strategies/plans for testing stay relevant to project objectives and software requirements.

Release Engineer/Manager:

- The RM ensures that the final software meets all baseline requirements and is coded properly for successful integration.
- The RM ensures that all documents are up to date according to the latest modifications and assembles data on the release metrics on the software.