
SOURCE CODE
MANAGEMENT
Dr Rubeena Doomun

Contents

LO1: Understand the importance of source code tools

LO2: Understand the benefits of source code management

LO3 : Apply source code management best practices

Introduction

Source code management (SCM) refers to the process of tracking changes to a source code repository.

SCM keeps track of a code base's history of changes and assists in resolving conflicts when merging updates from various contributors.

Issues

Developer 1 working on Feature 1 could make some edits and find out later that Developer 2 working on Feature 2 has conflicting edits.

Developer 1 would make edits and Developer 2 would unknowingly save over Developer 1's work and wipe out the changes.

Version Control

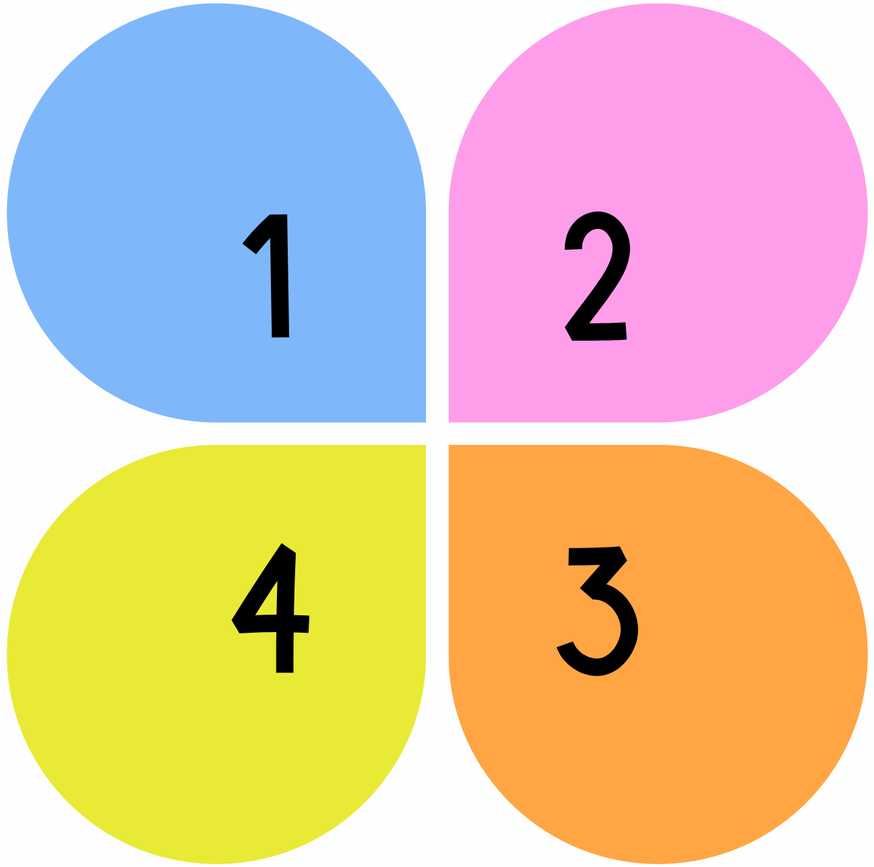
Version control protections were added to SCM to prevent work loss due to conflict overwriting.

These safeguards function by tracking each developer's modifications, detecting areas of conflict, and prohibiting overwrites

Benefits

Tracking all changes over time

Instantaneous rollback



Track of release versions

Parallel development through separate branches

SCM Best Practices

01 Commit often

02 Ensure you are working from latest version

03 Make detailed notes

04 Review changes before committing

05 Use Branches

06 Agree on Workflow

1. Commit often

Commit should be done on a regular basis to capture changes to a code base.

Each commit is a snapshot of the codebase that can be rolled back if necessary.

Many opportunities to rollback or undo work are provided by frequent commits.

2. Working from latest version

Multiple developers can quickly update a project using SCM. It's very easy for a local copy of the codebase to become out of sync with the global copy.

Before making any changes, be sure you git pull or retrieve the most recent code.

This will aid in the avoidance of conflicts throughout the merge process.

3. Detailed Notes

There is a log record for each commit. This log entry is filled with a message at the time of commit creation. It is critical to leave descriptive commit log entries that explain what's going on.

The "why" and "what" of the commit's content should be explained in these commit log messages. These log messages serve as the project's canonical history, leaving a trail for future contributors to follow.

4. Review changes before committing

SCM's offer a 'staging area'. The staging area can be used to collect a group of edits before writing them to a commit.

The staging area can be used to manage and review changes before creating the commit snapshot. Utilising the staging area in this manner provides a buffer area to help refine the contents of the commit.

5. Use Branches

Branching is a strong SCM feature that allows developers to construct a new development line.

Branches allow different developers to work on various lines of code in simultaneously.

When a branch's development is finished, it is merged into the main development line.

6. Agree on Workflow

SCMs provide relatively open-ended contribution techniques. It is critical for teams to develop shared collaboration patterns.

Patterns and techniques for branch merging are established by SCM workflows.

When it comes time to merge branches, if a team doesn't agree on a shared workflow, it might lead to inefficient communication overhead.

Activity Time

Thank you