

SCM Activities

1. CONFIGURATION IDENTIFICATION

For Configuration Identification of versions, throughout this project we will use Semantic Versioning 2.0.0 refers to a version naming convention for software. It is often abbreviated as SemVer. This convention helps communicate changes to software clearly and consistently.

Semantic Versioning (or SemVer) has three main parts to its version number: MAJOR, MINOR, and PATCH. Each part represents a different type of change to the software:

MAJOR version: Increased when changes that are incompatible with previous versions are made.

MINOR version: Increased when new features are added in a manner compatible with previous versions.

PATCH version: Increased for bug fixes while maintaining backwards compatibility

[Semantic Versioning 2.0.0](#)

Labels

Labels are identified based on their Issue Board and a status of the same board. Follow the template <Board>::<state>. Based on the <Board>, Boards are created. In certain cases, labels can only be created using <name> when it is not necessary to integrate it into a board or is a complement to the identification, such as **Bug**.

The priority labels **Priority Low**, **Priority Medium**, **Priority Critical** go beyond what was previously defined, but they could be created in the same way as the bug label, so we can consider it to be just their name.

Issues/UserStory

Issues or UserStory are identified by their name with the complement of unique and unambiguous identification managed by Gitlab, sequential numbering.

Epics

The use of Epics presupposes clear and objective identification. The use of epics presupposes the use of the template Epic#: . The must be the same as that assigned by Gitlab and the must be a short and objective name that portrays the epic.

Branches

All coding branches must be created from brance dev, they have already been created from master. Brands follow the Issue model. Please note that is always 2 digits (Example: 05).

SW Tests

The unit tests identified by the TS# model. With being the unambiguous numeric identifier of the issue (Example: 05) and being a sequential number for testing that issue. Examples: TS0503, TS1201.

Commits

Be sure to include the issue number, a clear and succinct name, and a comment that quickly and clearly specifies what was done in the change. If the Issue is not resolved in this commit, you should add WIP (work in progress) to the name of your commit. Make commits whenever there is a significant change. Example: #01 - Commit Example WIP.

Milestones

Each Milestone is identified by the Milestone#<number>. The that identifies the Milestone is numerically sequential based on its chronological order.

Milestone#1 - 17/12/2023

Milestone#2 - 29/12/2023

Milestone#3 - 15/01/2024

Sprints

Each Sprint is identified by the Sprint# template. The that identifies the Sprint is numerically sequential based on its chronological order. Each Sprint will last one week. Sprints end on Fridays, and another one begins immediately.

2. CONFIGURATION CONTROL

Boards

A board in GitLab is a valuable tool for managing and visualizing the progress of issues in a project, promoting more efficient and transparent collaboration between team members.

1. BackLog Board

The Backlog Board is an essential tool in agile methodologies, such as Scrum, which serves as a visual space to list, **BackLog Prioritize/Estimate** new tasks, issues or user stories in a project. It represents an organized set of pending work, where tasks come in, are refined and, when ready, move to the **BackLog Ready** phase to be included in future sprints or milestones.





2. Change Request Board

The Change Request Board manages change requests in a project, going through the states of "New", **Request Analysis**, **Request Approved** and **Request Rejected**. The Product Owner reviews and approves or rejects requests based on specific criteria. Approved and fully prepared requests are moved to the "Dev Board". If they are not approved, they are discarded, and if they are not detailed, estimated and prioritized, they go to the "Backlog".

This structured process guarantees controlled management of changes to the project.

If the change is a request from the client must have this tag **Request** Client Request .

3. Dev Board

The Dev Board plays a crucial role in agile project management by providing an organized visual representation of the progress of development tasks. This board covers several states, including "Open",  ,  ,  ,  and "Closed". Each state reflects a specific phase in the development lifecycle, from task selection to successful completion and closure.

How to Make a Change Request?

Changes cannot be made within an uncontrolled process.

So for submit a **Change Request** most follow the next steps:

1. Submit a Change Request in GitLab Issue Tracker, Change Request Board;
2. Product Owner reviews and decide if approve or reject;

If issue is already detailed, prioritized and estimated:

The Issue goes to the Dev board.

1. Plan sw tests;
2. Check-out and switch to dev branch;
3. Pull;
4. New branch for the issue;
5. Code;
6. Test locally;
7. Commit/Push;
8. Request Merge to Branch Dev;
9. Merge request approval by ... and pipeline = true;
10. Merge;
11. Push the merged Dev Branch;
12. Resolve the issue.

Else

The issue goes to the Backlog Board and must pass the steps of Backlog to be detailed, prioritized and estimated.

When the Issue is Ready:

The Issue goes to the Dev board.

1. Plan sw tests;
2. Check-out and switch to dev branch;
3. Pull;
4. New branch for the issue;
5. Code;
6. Test locally;
7. Commit/Push;
8. Request Merge to Branch Dev;
9. Merge request approval by ... and pipeline = true;
10. Merge;
11. Push the merged Dev Branch;
12. Resolve the issue.

3. CONFIGURATION STATUS AND ACCOUNTING

3.1 CM in Software Development Life Cycle

3.1.1 Software development life cycle

The Software Development Life Cycle (SDLC) is a systematic process used by software developers to design, develop, test, and deploy high-quality software. It includes various phases such as Planning, Feasibility Study, Requirements Analysis, System Design, Implementation (Coding), Testing, Deployment, and Maintenance. These phases can be executed sequentially or in an iterative and incremental manner, depending on the chosen model, such as the Waterfall model or Agile methodologies. The SDLC provides a framework for information system development, emphasizing quality, efficiency, and customer satisfaction.

The strategy to be used throughout this project will be the **Iterative and Incremental Model**.

The Iterative and Incremental Model is a software development model approach that combines the concepts of iteration and increment. This model aims to overcome some limitations of the traditional waterfall model by providing more flexibility and accommodating changes in requirements throughout the development process. Key features include repetitive cycles of development, continuous feedback, and the progressive construction of software in increments. The development cycle involves planning, implementation, testing, evaluation, and additional iterations based on feedback. The approach offers flexibility to adapt to changing requirements, delivers functional software after each iteration, and enables quick customer feedback. While advantageous for continuous delivery and adaptability, challenges include scope control and risk management.

For development phase we will use **Test-Driven Development (TDD)** is an approach to software development where tests are written before the actual code. This follows a cycle known as the Red-Green-Refactor cycle. In the Red phase, tests are created, deliberately causing them to fail. In the Green phase, the minimum code necessary is written to make the tests pass. Finally, in the Refactor phase, the code is improved without changing its external behavior.

TDD involves:

1. **Write Test Cases:** Create tests that define the expected behavior of the code.
2. **Run Tests:** Execute all tests, expecting them to fail initially.
3. **Write Code:** Implement the minimum code to pass the tests (Green phase).
4. **Run Tests Again:** Verify that all tests pass.
5. **Refactor Code:** Improve the code's structure without changing its behavior.
6. **Repeat:** Iterate through the cycle for each new functionality or improvement.

Benefits of TDD include early bug detection, improved code quality, easier maintenance, confidence in code changes, and enhanced collaboration. TDD is widely adopted in agile development and is considered a best practice in software development.

3.1.2 SCM PLAN in the Software development life cycle

1. Planning:

SCMP influences planning by establishing policies for configuration identification, version control, and change management, setting up a configuration framework from project inception.

2. Requirements Analysis:

Supports requirements analysis by providing guidelines for configuration identification and control, laying the foundation for tracking and managing changes to requirements.

3. System Design:

SCMP contributes to system design by ensuring that the configuration framework is considered in architectural development, enabling a controlled and traceable system design.

4. Implementation (Coding):

Is crucial for implementation, controlling code versions through commits and branches, enabling effective tracking of changes and ensuring code consistency.

5. Testing:

The SCMP supports testing by managing configurations in test environments, ensuring that correct software versions undergo testing and providing a controlled environment for issue identification. We will be implementing the Test-Driven Development (TDD) approach, planning tests before development. This integrated practice aims to enhance software quality, identify issues early in the development process, and streamline testing efficiency. It aligns with the overarching goals of the SCMP to maintain a controlled and well-managed environment for software development and testing activities.

6. Deployment:

Ensures a smooth deployment by controlling configuration identification and managing changes, ensuring the correct software version is delivered and maintaining system integrity.

In summary, SCMP permeates all SDLC phases, providing a robust framework for configuration identification, version control, and change management. This ensures a more controlled, traceable, and efficient software development throughout its life cycle.

3.2 Tools

The tools that we will use to develop this project will be:

- **Java 17** : a versatile, object-oriented programming language known for its portability, security, and wide adoption across diverse computing sectors.

[Official Documentation](#)

[Official Download](#)

- **Gradle 8** : a powerful and flexible tool that simplifies the process of building, testing, and distributing software, with a convention-based approach and a variety of features to make complex project management easier.

[Official Documentation](#)

[Official Installation](#)

- **JUnit 5**: a testing framework for Java that helps developers write and run automated tests. It offers annotations like @Test to mark test methods, assertions to check expected results, and support for parameterized testing. JUnit is widely used to promote the practice of unit testing in Java software development, contributing to code quality and reliability.

[Official Documentation](#)

[Gradle+JUnit 5 Documentation](#)

[IntelliJ Tutorial Gradle+JUnit 5](#)

- **Jacoco**: a code coverage analysis tool for Java. Its main function is to measure the amount of source code that is executed during the execution of automated tests.

[Official Documentation](#)

[Gradle+Jacoco Getting Started](#)

- **PMD**: a valuable tool for static source code analysis, offering developers an effective way to identify and fix potential problems, improve consistency, and maintain healthy coding standards in software projects. The PMD plugin performs quality checks on your project's Java source files using PMD and generates reports from these checks.

[Official Documentation](#)

[Gradle+PMD Getting Started](#)

- **IDE'S (IntelliJ/VSC)** : software tool that provides a comprehensive set of features for software developers, making it easier to write, test, and debug code.

[IntelliJ](#)

[Visual Studio Code](#)

- **SourceTree (optional)**: is a graphical interface to Git, making it easy to create, clone, and manage repositories. It offers a visual representation of branches, supports the Git Flow workflow, enables temporary storage of changes, provides a visual tool for code comparison and merging, integrates Git LFS for large file management, and is compatible with Windows and macOS. In short, Sourcetree simplifies the use of Git through a user-friendly interface.

[SourceTree](#)

3.3 Methodology

The agile methodology defined for this project is Scrum. Scrum is an agile framework used in software development and complex projects. It provides a framework for teamwork, project management, and delivering products in an iterative and incremental manner. It is widely adopted in software development, but its principles can be applied to a variety of contexts and industries. The framework promotes agility, transparency, and collaboration, contributing to the success of complex projects.

Throughout this project, we will have a Daily Scrum Meeting every day to track progress continuously. This meeting also serves as a planning session, where problems and other variable matters are discussed and controlled. Any deficiencies or restrictions in the system development process are actively sought, identified, and addressed during these daily meetings to continuously improve the overall process.

Regarding our Sprint ceremonies, the Sprint Planning Meeting is scheduled for Monday's. This meeting, held at the beginning of each Sprint, allows the team to collaboratively plan the work to be undertaken during the upcoming Sprint. It is a crucial session for setting clear objectives and priorities.

The Sprint Review and Retrospective will take place on Friday's at the end of each Sprint. During the Sprint Review, the team will showcase the work completed during the Sprint. Following this, the Retrospective is conducted to reflect on the Sprint's successes and areas for improvement, fostering continuous learning and adaptation.

This structured approach to Sprint ceremonies ensures a consistent rhythm for planning, execution, and reflection, enabling the team to deliver high-quality increments of work in a collaborative and efficient manner.

Pre-Game

- **Planning**

Includes the definition of the system to be developed. A Product Backlog list is created containing all requirements that are currently acquaintances, requirements are prioritized and the effort required for their implementation is estimated. In every iteration, the updated product is reviewed by the Scrum team to define the commitment for the next iteration.

To prioritize and estimate issues and user stories we will use Planning Pocker. For estimate we will use the Fibonacci sequence from 0 to 89, with 0 being a very simple task and 89 being a monstrous task where it will be necessary to divide it into sub-tasks, or have more people working in it.

The sequence will be 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.

- **High-level architecture/project**

The high-level design of the system includes planning the architecture with current items in the Product Backlog. A project review meeting is held to address of proposals for implementation and are taken decisions based on this review In addition, preliminary plans for the contents of releases are prepared.

Development Phase

Each Sprint includes the traditional phases of product development software: analysis, requirements, design, evolution and delivery. The architecture and design of the system evolve during the sprint development. A Sprint is planned to last between a week.

Post-Game Phase

Contains the closing of deliveries. The system is ready to be delivered, the requirements are complete, there are no more items to add. Includes tasks such as integration, testing of the system and documentation.