# PROJETO LDS

## *SCM PLAN*

Versão 2.0

12/01/2021

**Desenvolvido por:**

João Leocádio, Nº 8160523
Paulo Brito, Nº 8160279

# Índice

# 1. Introdução

Este trabalho prático, foi desenvolvido para a disciplina de Laboratório de Desenvolvimento de Software da Licenciatura em Engenharia Informática, recorrendo à matéria abordada na unidade curricular, com foco em práticas de garantia de qualidade de software, nomeadamente, *Software Configuration Management*. Ao longo deste projeto, seguiu-se uma abordagem baseada em *Continuous Integration*(integração continua), visando suportar e automatizar o melhor possível o processo de construção de software.

Deste modo, o projeto proposto consiste na elaboração de um software voltado para a vertente de livros. Através desta aplicação, será possível ao utilizador trocar, ver e seguir livros de vários outros utilizadores. Também será possível trocar mensagens através de um sistema de chat incorporado na aplicação.

Durante a implementação do software proposto, foi necessário definir roles, definir atividades de SCM a realizar, definir quando as atividades do SCM são realizadas em relação às atividades do projeto, definir ferramentas e recursos físicos/humanos necessários. Assim foram utilizadas inúmeras ferramentas, das quais o gitlab (repositório e controlo de versões), o framework ASP.NET Core, IntelijRider e Xamarin, etc.

# 2. Propósito

Este documento foi elaborado com os seguintes objetivos:

- Definir papéis (*roles*) e responsabilidades de cada membro do grupo.
- Definir todas as atividades SCM a serem executadas no contexto do projeto.
- Coordenar as atividades SCM com as atividades do projeto.
- Identificar as ferramentas e os recursos físicos e humanos necessários à execução do mesmo.

## 3. Âmbito

Neste plano, irá ser exposta a forma de como se vão distribuir as responsabilidades entre os membros da equipa, assim como a abordagem pretendida ao longo da evolução do projeto (estando este muito provavelmente sujeito a mudanças).

## 4. Termos chave

- SCM **–** *Software Configuration Management* (Gestão da Configuração do *software);*
- CM **–** *Configuration Management* (Gestão da Configuração);
- Issue – Tarefa a desenvolver/desenvolvida;

**Sprint** – Ciclos iterativos onde a funcionalidade é desenvolvida ou melhorada produzindo novos incrementos.

**Mainline** - Minimiza o *merging* e mantém o número de branchs minimo possível por desenvolvimento na Mainline.

**Branch** – Mecanismo que permite que parte da equipa realize uma determinada tarefa separadamente, sem comprometer o desenvolvimento da restante equipa.

# 5. Gestão SCM

O objetivo principal de um plano de gestão de configuração de software (SCM Management) é descrever todos os indivíduos envolvidos no projeto e também explicar de forma minuciosa os seus papéis por forma a assegurar uma boa organização garantindo assim o sucesso da aplicação.

## 5.1 Regras e responsabilidades SCM

**SCM Role: SCRUM Master**

**Quem? – Paulo Brito**

**Responsabilidades:**

- Aprova mudanças ao *SCM Plan.*
- Aprova a criação de novas issues.
- Aprova mudanças na abordagem de trabalho, alteração de *issues* e estabelecer os objetivos de sprint.
- Aprova *merge requests.*
- Organizar *meetings* com os restantes membros da equipa de modo a chegar a consenso relativamente a mudanças necessárias no projeto.

**SCM Role: DEV Team**

**Quem? João Leocádio, Paulo Brito.**

**Responsabilidades:**

- Desenvolver o código necessário para a resolução de uma *issue* ao qual foi previamente atribuído.
- Desenvolver os testes mediante os cenários de teste desenvolvidos ao código que desenvolveu de modo a verificar que o que implementou, vai de acordo com o pretendido.
- Quando concluir as duas tarefas anteriores deverá informar os responsáveis pela revisão do código e aguardar pela respetiva aprovação.
- Após obter a aprovação, deverá aí sim submeter o *merge request* e aguardar a decisão final tomada pelo SCRUM Master;

**SCM Role: SCM Team Leaders**

**Quem: João Leocádio**

**Responsabilidades:**

- Definir inicialmente *User Stories* e respetivas *Issues* em conjunto com o SCRUM Master.
- Sugere mudanças ao *SCM Plan*;
- Sugere alterações à própria documentação;
- Efetua o desenho dos testes (isto inclui ECP e BVA).

# 6. Atividades CM

Nesta secção, pretende-se mostrar as várias atividades de SCM, tais como, *Configuration Identification* (que tem como principal função clarificar como será realizada a numeração da versão do projeto), *Configuration Control* (pretende demonstrar as várias etapas do processo de controlo de alterações) e também *Configuration Status* (onde é definido o padrão SCM).

## 6.1 Identificação da configuração (*Configuration Identification*)

Todos os componentes e documentos começam o seu ciclo de vida com a versão 0. Deste modo, sempre que é fechado um issue, no qual o componente sofreu alterações, a sua versão é incrementada em 0.1. Por outro lado, e sempre que é concluída uma sprint que resulta numa release, a versão muda para o próximo número inteiro. (Exemplo: 0.5 -> 1.0).

Tendo por base que o número da versão seja um número com dois dígitos, parte inteira e parte decimal (por exemplo:1.1). Inicialmente o nosso software encontra-se na versão (0.0) e apenas altera o seu valor inteiro quando é implementada e testada uma funcionalidade que tenha dimensão considerável, ou seja, que atribua uma ou mais nova/s funcionalidade/s ao nosso software o que irá corresponder ao merge de uma *development code line* que irá corresponder a um *release* para o cliente.

Por outro lado, quando o *software* sofre alterações menos significativas (quando há correção de bugs, pequenas alterações na arquitetura nos requisitos e as mesmas são implementadas, ou é adicionada uma funcionalidade relevante à Development Code Line atual) existe apenas uma incrementação no número decimal que representa a versão atual (por exemplo: 0.1 ->0.2).

A nossa abordagem foi planeada de maneira a coincidir com a adição das ditas mudanças significativas com o *merge* da *development code line* atual (que ocorre por norma no fim de cada Sprint), de maneira a que, sempre que há um *merge* da *development code line* incrementa-se a versão por um dígito inteiro e faz-se um *release.*

## 6.2 Controlo da configuração (*Configuration Control*)

As *User Stories* são definidas numa reunião entre todos os membros da equipa. Cada *User Story*, será resolvida através de uma ou mais *issues*. Tanto *Issues* como *User Stories* serão reavaliadas ao longo do desenvolvimento do projeto e se necessário alteradas.

Cada *Story* terá um “estimate” e um "priority”. *Estimate* denota o esforço estimado para que cada *issue* seja resolvida, decidimos defini-lo através de tamanhos de roupas (S, M, X, XL), sendo que uma *Story* classificada com “S” será menos trabalhosa do que uma classificada com “M”.

*Priority* define o nível de prioridade atribuído à *User Story*. Para isso temos uma escala de 1 a 3, sendo 1 imprescindível ao funcionamento do software e 3 algo que não é fundamental para o funcionamento da aplicação, mas que se pretende implementar.

As *issues* são definidas no início do projeto e reavaliadas no fim de cada *sprint.* Alterações a *issues* terão de ser aceites por todos os membros da equipa antes de serem efetuadas.

Para cada issue, os SCM Team Leaders estabelecem *Test Cases*, posteriormente existe uma reunião com o grupo para acertar se as tabelas estão corretas. Deste modo, foi definido o seguinte:

- O membro responsável pela *issue* desenvolve o código;
- O membro responsável pela *issue* desenvolve os testes mediante o plano;
- *Add/commit/push* se o código passar nos testes;
- *Code review;*
- *Merge* caso o *commit* passe no *code review*;

A partir dessa *issue*, o membro responsável inicia um *branch* a partir da *active developoment line* que após estar devidamente concluído irá ser revisto pelo *code Reviewer.*

No fim da análise, é transmitida a informação ao *developer* que efetuará o *Merge Request* ou procederá às alterações do código mediante os resultados da análise.

Assim, o SCRUM Master é responsável por aceitar o *merge request* para a *active development code line*.

Após as funcionalidades atribuídas a esta *codeline* estarem devidamente desenvolvidas e testadas, será feito um *merge* da *active development codeline* para a *baseline*. Aqui temos uma alteração significativa no desenvolvimento do software, e por isso deve ser incrementada a versão no que toca ao número inteiro da mesma.

## 6.3 Estado de configuração (*Configuration Status*)

*Task Branch*: Para cada *issue* levantada será criada uma *branch* para completar o trabalho necessário. Utilizando este padrão para projeto garantimos, que antes de ocorrer o *merge* com a *mainline*, tudo se encontra funcional e de acordo com o plano.

*Mainline*: Após a conclusão de uma *issue* o código adicionado será *merged* com a *mainline* garantindo que o código mais recente está sempre presente na mesma.

*Release Line*:  Para realizar a manutenção de uma *Release version* será criada uma *branch* de maneira a isolar este trabalho do que ocorre na *mainline*.

*Codeline Policy*: Para cada *issue* será criada uma *branch* para a completar. O seu nome deve estar relacionado com a *issue* e o desenvolvedor deve utiliza la para completar o trabalho. Após a conclusão do processo a *branch* é apagada.

*Unit Test*: Após realização de alterações serão realizados testes unitários para garantir que nenhuma das alterações provoca erros no projeto.

# 7. CM no SLDC

## 7.1 SLDC: usado e como

O modelo de SLDC(ciclo de vida do software) utilizado no projeto é o *Agile*, em que o principal intuito é entregar o trabalho de forma incremental em colaboração com o cliente e também com os membros da equipa de desenvolvimento.

A existência da lista de *issues* (conhecida também *Product Backlog* em *SCRUM*) com as mesmas associadas a *Milestones* (ou *Sprints* com devidas *User Stories*) demonstra a integração entre o plano e a metodologia *SCRUM* que segue o Modelo Ágil.

Após cada *sprint*, existe uma fase de *review* com a equipa toda, e antes de ocorrer um *merge* existe um processo de validação referido anteriormente no ponto 6.2.

Como os requisitos podem mudar ao longo do projeto, é bom ter uma estrutura de revisão para novas *issues* que possam aparecer. Daí a necessidade do ponto 6.2 pois a alteração de mudanças não serve só para o que foi planeado, mas também para o que não foi.

O *Software Development Life Cycle* usado é *Agile*. Na metodologia *Agile*, a evolução do projeto é composta por *Sprints*, em cada uma delas passaremos sempre pelos mesmos processos que serão reavaliação de *Requirements* (as nossas *issues*, fazer uma análise das mesmas), o que pode até mesmo levar à descoberta de novos *requirements*, *Code* (desenvolver código para resolver as *issues*), Teste (testes ao código previamente desenvolvido) e *Review*, sendo que sempre que os Requisitos se alteram é necessário que se desenvolvam e testem novamente. A nossa Active *development code line* numa situação ideal apenas seria *merged* para a *main line* no final de uma *sprint*, como sabemos que isso pode não acontecer, esse *merge* apenas acontece quando um conjunto de funcionalidades são devidamente implementadas e testadas. Este conjunto é definido em reunião quando abrimos a *codeline.*

# 8. Ferramentas e Metodologia

## 8.1 Ferramentas

Intelij Rider: ide para desenvolvimento da aplicação;

GitLab (para controlo de versões):

No GitLab, encontra-se uma wiki com a documentação do projeto, e está previsto um ficheiro de configuração em Yaml para as Pipelines.

As pipelines do GitLab permitem garantir que o código que foi commited não quebra nenhuma das funcionalidades anteriormente criadas através do testing. Também garante que o código atual passa a fase de build. Isto permite-nos garantir a prática do Continuous Integration e Continuous Delivery.

O Git será utilizado como repositório local no computador de cada desenvolvedor, permitindo a submissão das alterações quando desejado e trabalho independentemente do repositório online.

## 8.2 Metodologia

A metodologia utilizada no projeto será SCRUM, partindo do modelo Ágil, em que o foco é entregar o produto de forma iterativa e incremental. SCRUM não se foca na documentação, mas sim na interação entre os envolvidos no projeto.

No scrum, existe o *Product Backlog* (*Open Issues* no GitLab) onde os requisitos atualmente conhecidos ficam até estarem realizados. Esta lista é constantemente atualizada á medida que o projeto avança.

À medida que o projeto se vai desenvolvendo através de Sprints que são ciclos iterativos, a funcionalidade é desenvolvida/melhorada produzindo assim novos incrementos.

A cada *Sprint* (*Milestone* no *Gitlab*) a equipa reúne-se para verificar o que foi criado e também para definir o que vai ser preciso para a Sprint seguinte.

## 8.3 Atualizações

De acordo com a evolução do projeto, o mesmo ocorrerá com este documento daí a existência desta tabela de versões.

| Version # | Implemented By | Revision Date | Approved By | Approval Date | Reason |
|---|---|---|---|---|---|
| 1.0 | João Leocádio, Carlos Meireles | 26/10/2020 | Manuel Garrido, Paulo Brito | 26/10/2020 | Plano Inicial |
| 1.1 | João Leocádio, Carlos Meireles, Manuel Garrido, Paulo Brito | 28/10/2020 | João Leocádio, , Manuel Garrido, Paulo Brito, Carlos Meireles | 28/10/2020 | Introdução;<br>SCM Management;<br>CM Activities |
| 1.2 | João Leocádio, Carlos Meireles, Manuel Garrido, Paulo Brito | 30/10/2020 | João Leocádio, , Manuel Garrido, Paulo Brito, Carlos Meireles | 30/10/2020 | CM in the Software Development Life Cycle;<br>Tools and Methodology |

| | | | | | |
|---|---|---|---|---|---|
| 1.3 | João Leocádio, Carlos Meireles, Paulo Brito | 02/11/2020 | Manuel Garrido | 09/01/2020 | Correção da ortografia |
| 1.4 | João Leocádio, Paulo Brito | 03/12/2020 | Manuel Garrido | 04/12/2020 | Redefinição das Roles e reestruturação do documento |
| 1.5 | João Leocádio, Paulo Brito | 04/12/2020 | Manuel Garrido | 04/12/2020 | Correção da ortografia |
| 1.6 | João Leocádio | 12/01/2021 | Paulo Brito | 12/01/2021 | Alteração do SCRUM Master e Dev Team |
| 1.7 | João Leocádio | 12/01/2021 | Paulo Brito | 12/01/2021 | Revisão Geral |