# PROJETO LDS

# HouseM8

## SOFTWARE CONFIGURATION PLAN

Version 1.0

03/11/2020

**Histórico de Versões**

| Versão # | Implementado por | Data De revisões | Aprovado por | Data de Aprovação | Motivo |
|---|---|---|---|---|---|
| *1.0* | *Jorge Moreira* | *03/11/2020* | *Miguel Costa*<br>*Samuel Cunha*<br>*Jéssica Coelho*<br>*Marcelo Carvalho* | *03/11/2020* | *Criação do plano SCM*<br>*Review do plano SCM* |
| | | | | | |
| | | | | | |

# Tabela de Conteúdos

# 1. Introdução

Este documento destina-se a descrever um conjunto de processos, políticas e ferramentas que ajudam/organizam o processo de desenvolvimento de software. Neste caso, o documento enquadra-se no âmbito de desenvolvimento do projeto, HouseM8, da disciplina de laboratório de desenvolvimento de software. Neste documento serão mencionados também a equipa de SCM e non-SCM (tipicamente a equipa de desenvolvimento, clientes, etc..) assim como os papéis e responsabilidades dos membros na equipa.

# 2. Identificação do documento

***SCMP02112020V1.0***

Descrição do identificador do projeto:

*SCMP* - SCMPlan

02-Dia do mês(dois dígitos iniciais)

11-Número do mês(dois dígitos seguidos dos dois dígitos iniciais)

2020-Ano de criação(quatro dígitos finais)

V2.1-Numero de versão do projeto(dois caracteres sendo "V" o identificador de versão através de sigla seguido do numero de versão "2" e numero de alteração ".0" )

# 3. Glossário

**SCM** – Software Configuration Management;

**CM** – Configuration Management;

**CCB** – Configuration Control Board;

# 4. SCM Management

## 4.1. Non-SCM members

### 4.1.1. Dev Team

Os membros da equipa de desenvolvimento são o Jorge Moreira, Samuel Cunha, Marcelo Carvalho, Miguel Costa e Jéssica Coelho. Quando um membro da equipa de desenvolvimento necessitar de fazer alterações ao código ou à documentação, estes devem de submeter uma *issue* no Gitlab ao ***CCB*** de modo a obterem aprovação. A *Dev* Team tem como por obrigação seguir as políticas de código estabelecidas pelo líder *SCM, (neste caso o líder SCM implementou o plugin PMD, logo terão de se guiar pelos reports e pelas regras default do plugin)* e o esquema de *branching* especificado na secção 7.

### 4.1.2. Customers

Um utilizador com estatuto de empregador ou trabalhador/mate, clientes da app, pretendem uma solução que simplifique a procura de trabalhos mais ágil e trabalhos rápidos, por isso tendem a recorrer aos nossos serviços. A equipa de desenvolvimento vai trabalhar de perto com alguns utilizadores para obter feedback de ambas as partes do sistema.

## 4.2. SCM Roles and Responsibilities

### 4.2.1. CM Team (Configuration management team)

A equipa **SCM** é composta pelo Jorge Moreira, Samuel Cunha, Marcelo Carvalho, Miguel Costa e Jéssica Coelho, responsáveis por fazer a manutenção da documentação e do código desenvolvido ao longo do projeto.

#### 4.2.1.1. CM Leader

O líder da equipa de SCM é o Jorge Moreira. O líder **SCM** define o **SCM** *plan*, estabelece a equipa de **CCB**, e define regras / políticas de escrita de código. É o responsável por atribuir responsabilidades aos membros da equipa. O líder da equipa de **SCM** é o responsável por *code review* e por organizar as reuniões de planeamento de sprints, para manter um desenvolvimento coerente e organizado do projeto. Mudanças significativas no código do projeto ou na documentação, necessitam de uma *review* e, no caso do código são necessárias *reviews* e testes por parte do Líder e dos outros elementos da equipa. **O CM Leader é também o Product Owner**.

### 4.2.2. Configuration Control Board (CCB)

O **CCB** é o responsável por avaliar e aprovar todos os pedidos de alterações quer de código, quer da documentação. O ***CCB*** é também responsável por decidir se a *baseline* está em condições de ter um upgrade e também decide se esta está pronta para ser testada. O processo de aprovação de uma *issue* pelo ***CCB*** está enquadrado na secção 5.2 deste documento.

#### 4.2.2.1. CCB Leader

Líder ***CCB***: Miguel Costa. O líder de CCB pode, de forma autónoma, aprovar pedidos de alterações aos componentes dos projetos e pode organizar reuniões para *reviews* e aprovar pedidos de alterações relativos ao projeto.

#### 4.2.2.2. CCB Members

Os membros do ***CCB*** são o Samuel Cunha, Marcelo Carvalho, Jorge Moreira, Miguel Costa. As atas de reunião são realizadas pela Jéssica Coelho, e aprovadas *pelos CCB members*.

# 5. SCM Activities

Nesta secção serão abordadas as atividades *SCM* a serem implementadas na realização deste projeto. Na secção 5.1 será explicado como irá ser feita inicialmente a numeração das versões do projeto e da documentação. A secção 5.2 mostra como serão feitas as alterações ao projeto, partindo de um pedido de alteração (*issue*) até a fase em que se encontra resolvido. Na secção 5.3 serão descritos os padrões SCM usados.

## 5.1. Configuration identification

A versões do projeto serão nomeadas da seguinte forma:

- Na fase inicial, o projeto será numerado como versão 0.
- Quando o projeto sofrer alterações mínimas, é adicionado um dígito ao número da sua versão, ou seja, se o projeto estiver na versão 0 e sofrer alterações mínimas, este passa à versão 0.1.
- Quando se trata de alterações / adição de funcionalidades em grande escala, a versão passa, por exemplo de 0.0 a 1.0.
- O software passa a ter o nome da versão "alpha" quando estiver numa fase inicial de testes e *pre-release*.
- Quando o software possuir funcionalidades já desenvolvidas, mas com bugs descobertos ou ainda por descobrir, passará a ser chamado de "Beta".
- Dentro da fase "Alpha" ou "Beta" poderão existir versões intermédias, como por exemplo "alfa - 1.0" ou "Beta - 1.1".
- Aquando de um release, o nome da versão será "Release – X" em que X é o número do release.

Os *configuration* itens são todos obtidos pelo repositório *Gitlab* da *ESTG* onde se encontram alojados os documentos e o próprio projeto. Todos os documentos contêm a sua própria tabela de versões nua página a seguir à capa. A documentação será disponibilizada na *wiki* do projeto no *Gitlab*. A documentação também poderá ser anexada à wiki, via snippet, isto é, anexam-se os documentos necessários em snippets e referenciam-se com os respetivos links na wiki.

## 5.2. Configuration control

A alteração ao projeto tem de ser feita de forma controlada. O processo de alterações ao projeto ocorre da seguinte forma:

1. É submetido um pedido na *Issue Tracker* no *Gitlab*;
2. O *CCB* faz uma *review* e aprova a *issue*;
3. A *issue* é então atribuída a um membro;
4. É feito um checkout da *branch master*;
5. É criada uma *branch* nova para a *issue*;
6. O código necessário é implementado;
7. Depois são feitos testes localmente;
8. De seguida, *commit* e *push*;
9. Verificar se os testes na pipeline passaram;
10. Depois é feita um *merge request*;
11. O *merge request* é revista em conjunto pelos membros do grupo e aprovado;
12. De seguida é feito o *push* da nova master;
13. Por fim, quando a *issue* for resolvida, esta é fechada.

14. Case necessário, o líder **SCM** faz a alterações na documentação ou seleciona outro membro para o fazer, se achar apropriado;

As *issues* de pedidos de alterações ao projeto podem diferenciar no nível de emergência, isto é, se for efetuado um pedido de alteração de código em que englobe um problema que ponha em causa a integridade de uma *feature* esta é classificada como urgente, com prioridade alta, isto é, poderemos estar a falar de um *hotfix* por exemplo. Caso seja um pedido de alteração em código cuja influência no programa seja mínima será colocada em baixa prioridade. Para isto poderá ser usado um *backlog* (especificado na secção 7).

## 5.3. Configuration Status and Accounting

***Mainline*** - Minimizar *merging* e manter o código estável em desenvolver numa *mainline*.

***Active Development Line* -** Manter código estável em rápida evolução, suficiente para ser útil criando uma *Active Development Line*.

***Private Workspace* -** Evitar que os problemas de integração causem distrações e/ou outros problemas, ao desenvolver num *Private Workspace*.

***Repository*** - Configurar um novo *Workspace* a partir de um repositório que contém tudo o que é preciso.

***Codeline Policy*** - Criar políticas de código para ajudar desenvolvedores a decidirem quando fazer o check-in do código numa *codeline* e quais procedimentos a seguir, antes de efetuar o check-in em cada *codeline*.

# 6. CM in the Software Development Life Cycle

O SDLC abordado neste trabalho é *agile*. Este método foca-se na entrega e na satisfação do cliente de modo a pretender sempre uma entrega rápida e continua de software funcional.

Neste modelo é muito importante que o cliente e a equipa de desenvolvimento trabalhem em conjunto durante o desenvolvimento do projeto para puderem adicionar alterações que possam surgir ao longo do desenvolvimento. Esta abordagem enquadra-se no CM quando aplicados os conceitos de *"Continuous Integration"*, *"Continuous Delivery"*, *"Continuous Deployment"* e *"Continuous Inspection"*, pois em cada incremento, o produto passa por fazes "build-test-relase", são sempre feitas inspeções e numa fase final do incremento o produto é declarado como *"production-ready"* e entregue.

# 7. Tools and Methodology

A metodologia ágil a ser usada no desenvolvimento deste produto é SCRUM. Ao seguir a metodologia *SCRUM* comprometemo-nos a desenvolver o software de uma forma iterativa, usando *sprints*. No caso deste trabalho, os sprints vão ser mencionados no *Gitlab* como *milestones*. Estes *sprints* devem ser iniciados depois de um *sprint planning*, uma reunião entre os membros do grupo que definem um *backlog* priorizado para a equipa de desenvolvimento implementar no trabalho. *Sprint review* também irá ser feito de modo a perceber o que foi feito ao longo do tempo até a um determinado sprint, o que se deve alterar / corrigir, como por exemplo *bugs* ou *features*, que pode ser útil para englobar no planeamento do *sprint* seguinte.

Os *backlogs* são uma lista priorizada de trabalho, cuja pode conter *user stories*, *epics* (conjuntos de *user stories*), correções de *bugs* ou novas *features* que permite aos desenvolvedores saberem quais as tarefas mais importantes a cumprir. Caso o *backlog* cresça demasiado, há a necessidade de reunir os membros do grupo para reestruturar e, possivelmente dividir o *backlog* de modo a colocar tarefas para outro *sprint*. Quando o *backlog* é feito, é importante que este fique de acordo com o desenvolvimento feito ao longo do tempo, logo são necessárias *reviews* periódicas ao *backlog*. Epics, user stories, correções de bugs e outros devem de ser todos declarados no issue tracker do Gitlab. As user stories devem de ser representadas no seguinte formato:

**"As a [person], I [want to], [so that]."** Exemplo:

| **Title:** | **Priority:** | **Estimate:** |
|---|---|---|
| **As a** *<type of user>*<br>**I want to** *<perform some task>*<br>**so that I can** *<achieve some goal>* | | |
| **Acceptance criteria**<br>**Given** *<some context>*<br>**When** *<some action is carried out>*<br>**Then** *<a set of observable outcomes should occur>* | | |

*Figura 1 - User Story*

É necessário definir o critério de aceitação da *user story*, ou seja, o que esta deve de fazer e em que contexto. A user story poderá ter como título: **"As a [person], I [want to], [so that]"**.

Para controlo de versões, irá ser usado o *Gitlab* da *ESTG*. Não só servirá como repositório de projeto, mas também vai servir para anexar toda a documentação e gerir o projeto usando *SCRUM*. Também vão ser usadas pipelines no *Gitlab* para executar jobs como *build* e *tests* de forma automatizada a cada *commit*.

Será usada uma estratégia de *branhcing* baseada na estratégia de *gitflow* como mostra o exemplo na figura seguinte:

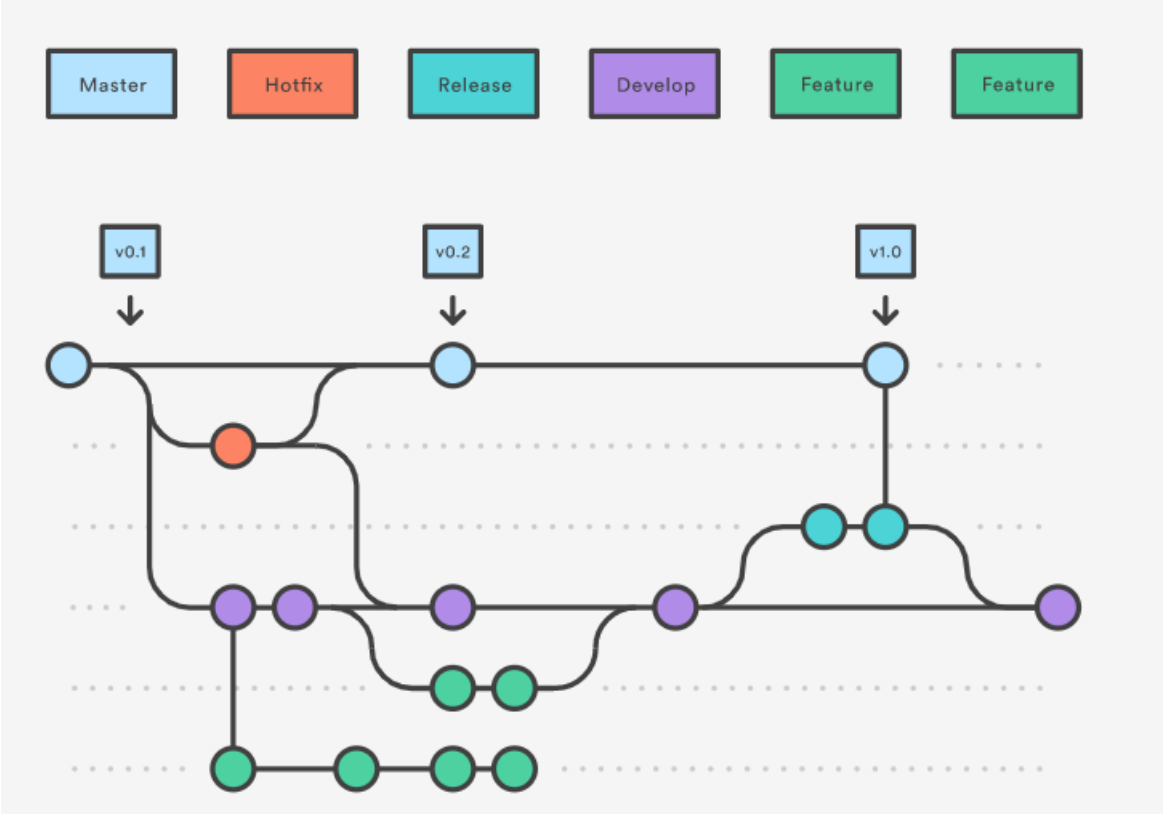


*Figura 2 - Branching gitflow – retirado de : https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow*