

---

# Gerenciamento de Configuração de Software 2ST

***Jonata William de Arruda***

*Universidade Tecnológica Federal do Paraná - Câmpus Cornélio Procópio*

---

**P**lano de Gerenciamento de Configuração de Software da 2ST visa especificar o processo para controle de versões, atualizações, distribuições e correções (bugs) relacionado aos produtos de software Klassic da empresa 2ST. Incluso a este documento um guia passo a passo (guideline) de integração a novos desenvolvedores, assim ambientando de forma prática e eficaz as rotinas e processos do desenvolvimento do produto.

*13 de dezembro de 2017*

## Lista de figuras

- |   |  |   |
|---|--|---|
| 1 | Diagrama exemplo de Controle de Versão . . . . .                         | 6 |
| 2 | Waffle.io - Exemplo de Controle de Mudança utilizando a ferramenta . . . | 7 |

## Lista de tabelas

- |   |  |   |
|---|--|---|
| 1 | Ferramentas utilizadas para Controle de Gerenciamento de Software na 2ST | 5 |
|---|--|---|

# Sumário

<b>1</b>	<b>Introdução</b>	<b>4</b>
<b>2</b>	<b>Estado atual</b>	<b>4</b>
<b>3</b>	<b>Requisitos</b>	<b>4</b>
<b>4</b>	<b>Ferramentas</b>	<b>5</b>
<b>5</b>	<b>Controle de Versão</b>	<b>5</b>
5.1	Repositório . . . . .	5
5.2	Baseline . . . . .	5
5.3	Nomenclatura de versões . . . . .	5
5.4	Diagrama . . . . .	6
5.5	Padrão de Commit . . . . .	6
<b>6</b>	<b>Controle de Mudanças</b>	<b>7</b>
<b>7</b>	<b>Processos</b>	<b>8</b>
7.1	Papéis . . . . .	8
7.2	Novas Distribuição . . . . .	8
7.3	Correções de bugs . . . . .	9
7.4	Nova Funcionalidade . . . . .	9
7.5	Nova Versão . . . . .	10
<b>8</b>	<b>Auditoria</b>	<b>10</b>
<b>9</b>	<b>Guideline</b>	<b>11</b>
9.1	Ambiente de desenvolvimento . . . . .	11
9.1.1	Dependências . . . . .	11
9.1.2	Instalação do Git . . . . .	11
9.1.3	Clonar repositório do software Klassic . . . . .	11
9.1.4	Instalando Java . . . . .	11
9.1.5	Instalando Maven . . . . .	12
9.1.6	Instalando SGBD e configurando Banco de Dados . . . . .	12
9.1.7	Próximo passo . . . . .	12
<b>10</b>	<b>Referências bibliográficas</b>	<b>13</b>

# 1 Introdução

O Gerenciamento de Configuração de Software é parte da Engenharia de Software e contém um conjunto de atividades de apoio ao desenvolvimento onde permite que as mudanças inerentes sejam absorvidas pelo projeto de maneira controlada, mantendo a estabilidade na evolução do software, pois mudanças durante o desenvolvimento são inevitáveis.

Sendo assim, o objetivo deste documento é especificar o processo de Gerência de Configuração de Software da empresa 2ST, referente o produto Klassic, afim de atender o mercado que busca por maior qualidade e produtividade no processo de desenvolvimento. Foram criados processos para controlar novas versões, atualizações, correções e distribuições, além de um guia para integração de novos desenvolvedores.

## 2 Estado atual

A empresa 2ST atua no ramo de software. Seu perfil de investidores é alinhado com o mercado e atende as expectativas. Possui carteira de clientes generosa. A equipe de desenvolvimento é composta por 10 integrantes. Possui o produto de software Klassic o qual foi desenvolvido por um arquiteto de software que não está mais na organização.

Atualmente existem clientes que necessitam de customizações. A customização de um cliente pode não atender outro e há clientes que não necessitam de customizações. O software poderá ser otimizado ou novas funcionalidades poderão ser inseridas: A otimização/nova funcionalidade poderá ser incorporada em todos os clientes. Novas versões poderão ser lançadas.

## 3 Requisitos

- Desenvolvido em Java.
- Utiliza JPA.
- Utiliza PostgreSQL.
- Ferramenta IDE NetBeans 8.
- Utiliza de maneira restrita a geração de código.
- Diagramas desenvolvido com Astah.

## 4 Ferramentas

As ferramentas utilizadas para o Gerenciamento de Controle de Software adotados pela empresa 2ST são:

**Tabela 1:** *Ferramentas utilizadas para Controle de Gerenciamento de Software na 2ST*

Tipo de Ferramenta	Nome
Controle de Versão	GitHub
Controle de Mudança	Waffle.io

## 5 Controle de Versão

Para controlar as versão dos produtos de software é utilizado o GitHub. Através da ferramenta é possível gerenciar todas as mudanças do produto, conferir o autor de cada alteração, criar tags para identificação de versão e rastreabilidade, padronizar mensagens de commit, realizar buscas por datas, tags, usuários, versões, entre outros que possam ajudar em uma auditoria, levantamento de métricas e demais necessidades relacionadas ao desenvolvimento do software.

### 5.1 Repositório

Para cada produto de software será mantido uma branch "master" onde todo o desenvolvimento do mesmo é contido.

Em caso de customização específica por um cliente, esta não sendo aproveitada na versão do produto principal, "master", é gerada uma nova branch com o nome do cliente.

### 5.2 Baseline

Como mencionado anteriormente a branch principal de cada produto de software terá como rótulo "master", está é a baseline padrão onde manterá o controle de versões seguindo as nomenclaturas adotadas.

Em caso de branch gerada ao cliente, esta apresentará a nomenclatura padrão acrescida de uma quarta casa decimal para controle de customização, assim, esta branch continua sendo monitorada pela versão "master" podendo receber suas atualizações, e é possível controlar as versão de customização do cliente específico.

### 5.3 Nomenclatura de versões

Será utilizado da seguinte forma:

**Version** : Utilizado para informar qual versão encontra se o produto.

**Function** : Alterado apenas quando novas funcionalidades são incorporadas no produto.

**Bugs** : Alterado quando uma correção faz necessário.

**Customize** : Alterado sempre que uma customização específica de um cliente é feita. Utilizada apenas em branch de clientes e não na "master".

Exemplo: *master 1.1.1* ou *cliente A 1.4.10.3*

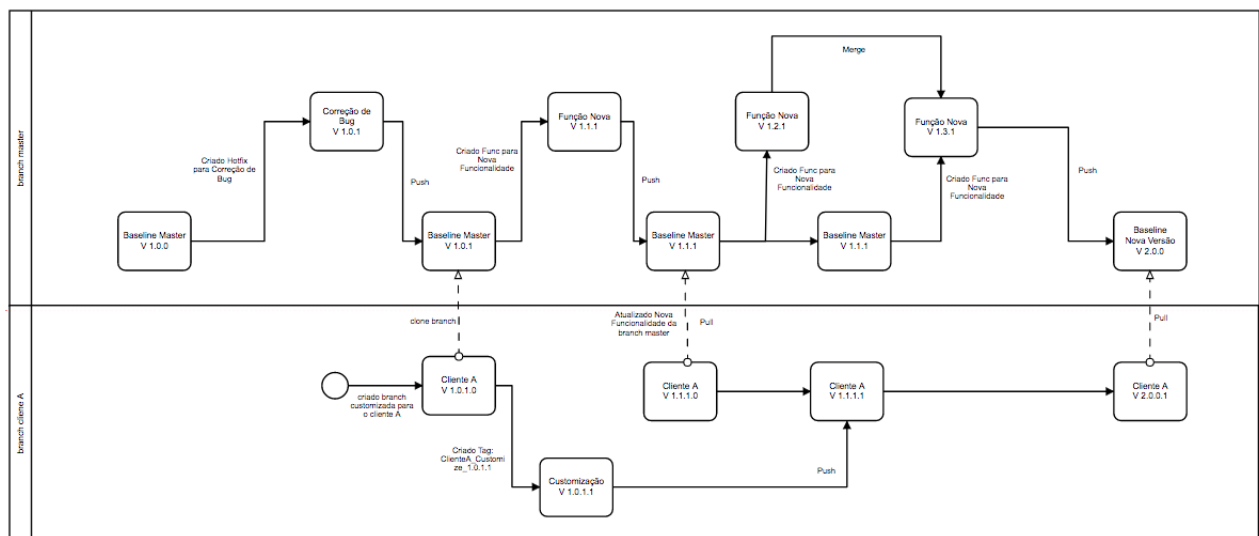
## 5.4 Diagrama

Abaixo é apresentado um exemplo em forma de diagrama do controle de versão utilizando a nomenclatura adotada.

Pode-se notar a branch "master" sofrendo alterações tanto de novas funcionalidades (Function) quanto de correções (Bugs), ao mesmo tempo uma customização foi necessária sendo criada uma nova branch para o cliente "A". Esta foi clonada da branch "master" com a versão atual. Após a customização, esta branch foi acrescida de em sua nomenclatura (Customize) onde passou a ser rastreada através deste número.

Por fim, a branch "master" incorporou mais algumas funcionalidades desenvolvidas e fechou seu ciclo, apresentando uma nova versão da branch, o que logo foi atualizado na branch do cliente A.

**Figura 1:** Diagrama exemplo de Controle de Versão



## 5.5 Padrão de Commit

Os "commit" devem seguir um padrão afim de tornar mais fácil a compreensão e integração por parte de novos colaboradores, deixando claro e resumido a alteração aplicada.

**Nova Distribuição** : Deve ser iniciado com a palavra "Nova Distribuição" seguindo de um espaço e depois "-", mais um espaço e o resumo objetivo da ação executada, neste caso o nome da nova distribuição e o cliente. Exemplo: "Nova Distribuição - sistema de carro cliente Volvo"

**Correções — Bugs** : Deve ser iniciado com a palavra "Correção"seguido de um espaço e depois "-", mais um espaço e o resumo objetivo da ação executada, neste caso o que foi corrigido. Exemplo: "Correção - Mascara do campo CEP "

**Nova Funcionalidade** : Deve ser iniciado com a palavra "Nova Funcionalidade"seguido de um espaço e depois "-", mais um espaço e o resumo objetivo da ação executada, neste caso o que seria a nova funcionalidade. Exemplo: "Nova Funcionalidade - Valida CPF"

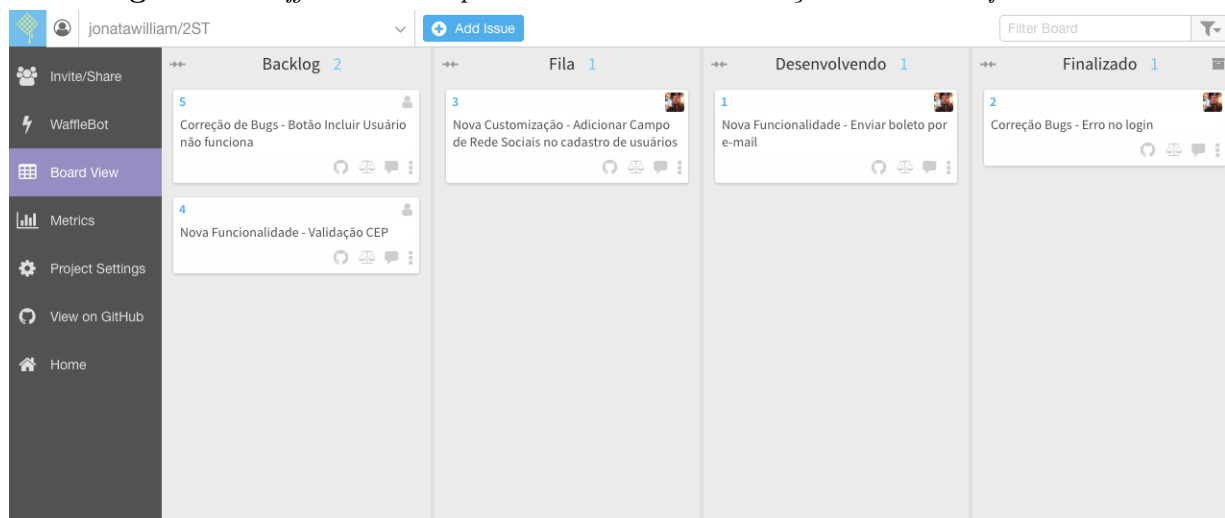
**Nova Versão** : Deve ser iniciado com a palavra "Nova Versão"seguido de um espaço e depois "-", mais um espaço e o número da nova versão respeitando a nomenclatura. Exemplo: "Nova Versão - 2.0.0"

## 6 Controle de Mudanças

A ferramenta Waffle.io proporciona uma integração com o GitHub o que tornar mais atrativo e eficiente na questão de rastreabilidade e o progresso do desenvolvimento.

Utilizando as Issues gerada para controlar todos os estados de produção da mesma, ou seja, toda movimentação do "card" até sua conclusão, desta forma, foi adotada a ferramenta Waffle.io para controlar as mudanças do produto de software da empresa.

**Figura 2:** *Waffle.io - Exemplo de Controle de Mudança utilizando a ferramenta*



## 7 Processos

### 7.1 Papéis

A seguir é descrito a competência que cada membro da equipe deve seguir baseado em seu cargo, estes seguindo uma padrão mencionado na referência [?].

Os papéis e suas atribuições são:

**Analista** : Representa os interesses do cliente e dos usuários finais recolhendo informações dos stakeholders para entender o problema a ser resolvido, capturando os requisitos e definindo suas prioridades.

**Desenvolvedor** : Responsável por desenvolver uma parte do sistema, incluindo a construção de seu design de forma que ele atenda a arquitetura e possivelmente a prototipagem da interface de usuário, e então implementar, executar o teste de unidade, e integrar os componentes que são parte da solução.

**Gerente de Projeto** : Conduz o planejamento do projeto, coordena as interações com os stakeholders e mantém a equipe de projeto focada em alcançar os objetivos do projeto.

**Cliente** : Grupos de interesse cujas necessidades devem ser atendidas pelo projeto. E um papel que deve ser executado por qualquer pessoa que é, ou será, afetada materialmente pelo resultado do projeto.

### 7.2 Novas Distribuição

Uma nova distribuição só é gerada em duas situações:

1. Caso seja um novo projeto.
2. Necessidade de uma customização de alguma versão para um cliente específico, e somente se esta customização não se aplicar de forma genérica a atender todos os clientes da versão principal (master) do produto. Sendo assim, é gerado uma nova "branch" do produto a partir da versão atual, e ela é acrescida de uma nova casa decimal, onde será controlado o número de customizações que sofreu.

Para chegar a tal ponto, deve ser analisado toda a coleta de dados do cliente feita pelo Analista, onde será repassada ao Gerente de Projeto que por sua vez conduzirá e delegará as atividades aos desenvolvedores afim de atender esta nova distribuição.

Novas distribuições de produtos existentes são evitadas ao máximo afim de manter uma única versão do produto o que facilita sua continuidade e não desperdiçando tempo e desenvolvimento em várias branch, porém sempre que não houver como evitar, uma nova distribuição será criada para atender o cliente.



## 7.3 Correções de bugs

O Analista deve analisar o problema relatado pelo cliente / usuário, e verificar se é coerente e verdadeiro, pois alguns relatos de bugs pode ser apenas a má utilização ou falta de conhecimento do mesmo. Caso o Analista identifique que realmente é um problema no produto, o mesmo reporta ao Gerente de Projetos que irá inserir na fila de correções (backlog) com suas devidas prioridades.

Após este passo e seguindo o padrão de nomenclatura, toda correção deve seguir os seguintes passos:

1. O Desenvolvedor seleciona o bugs a ser corrigido e aciona o processo de correção.
2. Abre uma novas hotfix de correção, onde será mantido o número de versão seguindo do número de função e incrementado o número de correções, assim seguindo a nomenclatura. Para este processo, o mesmo deve autenticar a abertura da hotfix com seus dados de login do GitHub que está vinculado a conta da empresa e habilitado a realizar modificações, assim será possível rastrear o que o desenvolvedor realizou neste hotfix.
3. Assim que corrigido e testado, o mesmo realiza um "commit" seguindo as regras de mensagem para "commit" estabelecidas neste documento.
4. Após o "commit" realizado, realiza o "push" para incorporar a branch principal (master) esta correção, e a mesma estar disponível para atualizações nos clientes que utilizam o produto e demais distribuições, principalmente no cliente que relatou o ocorrido.

## 7.4 Nova Funcionalidade

Todas a necessidades que o cliente ou o usuário do produto relata, ou seja, coisas que poderiam ter no produto, que poderiam ser de outro jeito e etc, esta direta ou indiretamente, é tratada, analisada e filtrada afim de que o Analista leve ao Gerente de Projeto e equipe de Desenvolvimento e possam discutir sobre o grau de importância e complexidade do mesmo.

Este processo sempre feito para que possam discutir ver se tal funcionalidade não existe, se existe e poderia melhorar ou realmente é algo novo, porém leva sempre em consideração a idéia de construir algo que atenda a todos os clientes dentro da versão principal (master) evitando novas distribuições.

Com esta análise feita, segue o processo de desenvolvimento parecido com o de correção:

1. O Desenvolvedor seleciona a issue de novo funcionalidade a ser desenvolvida.
2. Abre uma novas func para novas funcionalidade, onde será mantido o número de versão seguindo e incrementado o número de função e e mantido o número de correções, assim seguindo a nomenclatura. Para este processo, o mesmo deve autenticar para realizar a abertura da func com seus dados de login do GitHub que está vinculado a conta da empresa e habilitado a realizar modificações, assim será possível rastrear o que o desenvolvedor realizou neste func.

3. Assim que desenvolvido a nova funcionalidade e testado, o mesmo realiza um "commit" seguindo as regras de mensagem para "commit" estabelecidas neste documento.
4. Após o "commit" realizado, realiza o "push" para incorporar a branch principal (master) esta nova funcionalidade, e a mesma estar disponível para atualizações nos clientes que utilizam o produto, e demais distribuições.

## 7.5 Nova Versão

Sempre que novas funcionalidades são acumuladas e estas acabam não sendo atualizadas, e/ou é definida um ciclo de atualizações e este chega ao fim, uma versão é fechada e dá início a uma nova versão do produto. Isto também para fins marketing.

Para criar uma nova versão, é seguido os passos a seguir:

1. Cria-se uma nova tag com a versão nova, respeitando a nomenclatura adotada neste documento. Exemplo: Versão anterior 2.3.4. Nova Versão - 3.0.0
2. Realiza um commit com todas as func e hotfix finalizadas dentro do ciclo específico para esta nova versão. Utilize o padrão de commit estabelecido neste documento.
3. Realize a atualização desta nova versão tornando disponível para atualização dos clientes.
4. Em caso de distribuições diferentes, faz-se necessários a atualização (merge quando necessários) dos mesmos.

## 8 Auditoria

Este processo é feito utilizando as ferramentas de Controle de Versão e Controle de Mudança, uma vez que ambas oferecem recursos de rastreabilidade.

No Waffle.io é possível rastrear todas as Issues criadas, sabendo desde a hora e data, quem criou, quando foi desenvolvida ou alterado o card de status, qual data de conclusão, o tempo gasto e mais alguns outros itens.

Com o GitHub, é possível rastrear desde a abertura de uma func, hotfix, criação de branch, tag. Através das tags é possível saber quando foram criadas novas versões. Por fazer necessário o login de cada usuário do GitHub, principalmente para participar do diretório dos projetos da empresa, é possível rastrear todas as interações dos usuários, saber pelo log de cada modificação qual usuário que fez, quando e hora, o que foi feito entre outros dados. É possível também rastrear modificação seguindo os commits, uma vez que utilizando os padrões adotados por este documento, cada nova função, correção de bugs, nova distribuição e versão do produto torna mais fácil de achar e filtrar dentre os vários commits realizados.

## 9 Guideline

Este passo a passo referece a integração do novo colaborador afim do mesmo conseguir instalar o ambiente de desenvolvimento que a empresa adota e posteriormente conseguir colocar em prática este documento de Gerenciamento de Configuração de Software, que trás todos requisitos e passos necessários para compreensão e fluidez do processo de desenvolvimento.

### 9.1 Ambiente de desenvolvimento

#### 9.1.1 Dependências

Ubuntu 16.04 LTS

#### 9.1.2 Instalação do Git

Atualize seu OS:

```
sudo apt update
```

Comando para instalar o Git

```
sudo apt-get install git
```

#### 9.1.3 Clonar repositório do software Klassic

Git Clone:

```
git clone https://github.com/jonatawilliam/2ST/klassic.git
```

Acesse o diretório clonado:

```
cd klassic
```

Acesse a branch de desenvolvimento desejada, ex:

```
git checkout 1.0.0
```

#### 9.1.4 Instalando Java

Insira os repositórios do Java.

```
sudo apt-get install python-software-properties sudo add-apt-repository ppa:webupd8
```

Atualize o OS.

```
sudo apt-get update
```

Insira os comandos no terminal para instalar a máquina Java

```
sudo apt-get install oracle-java8-installer
```

### 9.1.5 Instalando Maven

Instale o Maven:

```
sudo apt-get install maven
```

Instale as dependências do projeto através do Maven:

```
mvn dependency:go-offline mvn install
```

```
mvn dependency:go-offline mvn install
```

### 9.1.6 Instalando SGBD e configurando Banco de Dados

Instale o SGBD PostgreSQL:

```
sudo apt-get install postgresql postgresql-contrib
```

Cria a database:

```
sudo psql -U postgres -c 'CREATE DATABASE klassic;'
```

Inicialize o banco:

```
sudo psql -U postgres -d aula -a -f Scripts/Database.txt
```

Certifique-se que dentro do arquivo `src/main/resources/META-INF/persistence.xml` as propriedades `jdbc.user` e `jdbc.password` estejam corretas de acordo com o seu banco de dados.

### 9.1.7 Próximo passo

A partir deste ponto, segue-se todos os processos descritos anteriormente neste documento para poder iniciar o trabalho de desenvolvimento do produto Klassic.

No repositório do GitHub está descrita a Guideline com todos os pontos-chaves deste documento no que se refere ao papel do desenvolvedor, assim possibilitando ao novo colaborador ter mais uma fonte de informações sobre como proceder com o seu trabalho e ao colaborador que faz parte da empresa, consultar sempre que necessário.

## 10 Referências bibliográficas