Plano de Gerenciamento de Configuração do Software Free Breeze

Histórico de Revisão

Data	Versão	Responsáveis	Mudança realizada
24/11/2020	0.0.1	*Leandro Peres Geovani Alves*	Criação do Documento de Configuração de Software
24/11/2020	0.0.2	*Geovani Alves*	Revisão conteudística
25/11/2020	0.1.0	*Leandro Peres*	Adequações e esclarecimentos
26/11/2020	0.1.1	*Leandro Peres*	Revisões e correções linguísticas.

1 Introdução

O Plano de Gerência de Configuração do ***Free Breeze***, aqui expressado, apresenta todas as tarefas do Gerenciamento de configuração e mudanças no projeto, sendo para garantir a sua integridade e manter o domínio das mudanças ocorridas durante o desenvolvimento.

Neste documento, detalha-se toda infraestrutura utilizada nesse projeto. Assim sendo, são definidos itens como: uma padronização para os documentos, uma ferramenta de controle da instalação da aplicação, preparo do ambiente de desenvolvimento de forma otimizada, integração ferramenta de controle de testes e por fim relatar os resultados da implantação da gerência de configuração de software. Para isso, devem ser seguidas diretrizes apresentadas em tópicos pelo plano de gerência.

1.1 Resultados esperados

Com a implantação deste plano, espera-se diminuir as ocorrências de problemas na configuração da aplicação em outros ambientes, além de garantir maior qualidade no desenvolvimento de testes da aplicação, assegurando que as entregas de software aceitas estejam no nível aceitável definido pela equipe de gerência.

1.4 Definições, Acrônimos e Abreviações

Termo	Significado
Baseline	Conjunto de itens de configuração que conquistaram um estado comprovado de estabilidade.
GCS	Gerência de Configuração de *Software*
CR	Solicitação de Mudança (*Change Request*)

2 Organização

2.1 Identificação dos documentos

Todos os artefatos produzidos devem manter o nome seguindo o título do artefato sem abreviação. Ex: `Plano de Gerência de Configuração`.

2.2 Versão dos Artefatos

O versionamento dos artefatos será feito por tabela de Histórico de Versão no próprio documento. Os artefatos podem ser desenvolvidos na ferramenta *Google Drive*, ou diretamente no *Notion*.

2.3 *Baseline* do Projeto

A baseline será gerada a cada fim de sprint do projeto, tendo assim um incremento no número da versão do sistema `1.0.0`, no caso de atualizações e correções de erros em versões passadas, deve-se utilizar o código da versão anterior sendo que o dígito após o "ponto" deve ser acrescido um `1.1.0`. Em linhas gerais, segue-se o modelo Versionamento Semântico 2.0.0.

2.4 *Branches*

Essa seção determina as políticas de *branches* utilizada no processo.

- Primordialmente, deve-se ater ao fluxo de trabalho Gitflow;
- Haverá duas *branches* principais: **main** e **develop**;
- A `main` manterá a versão entregável do projeto;
- A *develop* será utilizada para a equipe de gestão avaliar os *commits* e coletar as métricas;
- O incremento à master, ocorrerá através de *pull request* advindo da *branch development* e será de responsabilidade exclusiva da equipe de gestão;
- O incremento à *development* ocorrerá através de *pull request* advindo das *branches* de desenvolvimento, onde a aceitação ou recusa do *pull request* será de responsabilidade da equipe de gestão;
- Só poderão ser criadas *branches* de desenvolvimento apenas se forem relacionadas a casos de usos à serem implementados ou para mudanças no sistema;
- O nominação das *branches* de desenvolvimento devem seguir o modelo do Gitflow, sendo, portanto: `Hotfix/<nome da história de usuário>`, para concertos explícitos, e `Feature/<nome da história de usuário>`, para implementações de novas funcionalidades.

4. Commits

4.1 Comentário do *commit*

Os nomes dos commits devem ser relevantes ao conteúdo inserido ou retirado do repositório. Devem ser inscritos em língua portuguesa e conteúdo significativo, suficiente para identificação do que e onde houve alteração de código.

> *Exemplo:* `git commit -m "<titulação simples do commit>"`

4.2 Política de Commits

Quando houver pareamento em um *commit* deve-se usar:

> *Exemplo:* `git commit --author user_name --email user_email`

Em que o *user_name* é o nome do usuário com o qual está pareando, e *user_email* para o email respectivo ao mantenedor.

> *Exemplo:* `git commit --author Leandro --email leandro@peres.dev`

O commit deverá estar associado diretamente a um membro da equipe. No caso de pareamento, o commit deve possuir a assinatura de ambos. O *commit* deverá ter uma descrição para indicar o que foi solucionado.

Outra forma de editar um *commit* seria o uso do comando:

> *Exemplo:* `git commit -s`

Este comando abrirá o editor de texto e deverá seguir o seguinte padrão: O título para o commit, o nome da tarefa em desenvolvimento, uma breve descrição para o que foi feito naquele commit e o `Signed-off-by: "Nome do Usuario" example@email.com` mesmo se o commit foi dado apenas por uma pessoa. Ao fazer pareamento ou dojo deve haver o Signed-off-by com todos os contribuidores.

4.3 Solicitação de Mudanças

Deve seguir o tópico 4.1 e no conteúdo do *commit* e deve haver o código da *issue* gerado pelo Github.

> *Exemplo:* `git commit -m "ISSUE#X <titulação curta da história do usuário>"`

5. Ferramentas

A tabela abaixo descreve as ferramentas que serão utilizadas no desenvolvimento do projeto Free Breeze.

Ferramenta	**Descrição**	**Observações**
Travis CI	Integração Contínua	- Unity3D Travis CI - **Não** implementado.
Git	Controle de Versão	
GitHub	Serviço Web Hosting Compartilhado	
C#	Linguagem de Desenvolvimento	ver.: ***7.x***
Unity 3D	Motor gráfico e física	ver.: ***2019.4 LTS***
SemVer	Semântica de versionamento	A Semantic Version Library for .Net

6.1 TravisCI

O Travis CI proporciona um ambiente de build padrão e um conjunto padrão de etapas para cada linguagem de programação. É possível personalizar qualquer passo neste processo através do arquivo `.travis.yml`. O Travis CI usa o arquivo .travis.yml na raiz do repositório para tomar conhecimento sobre o projeto e como a build deve ser executada. O `.travis.yml` pode ser escrito de forma minimalista ou detalhado. Alguns exemplos de que tipo de informação o arquivo `.travis.yml` pode ter:

- Que linguagem de programação o projeto usa;
- Quais comandos ou scripts devem ser executados antes de cada compilação (por exemplo, para instalar ou clonar as dependências do projeto);
- Qual comando é usado para executar o conjunto de testes.

6.1.1 Motivação

As principais motivações para uma possível implementação do Travis CI:

- Gratuito para repositórios públicos;
- Ser um serviço de Integração Contínua na nuvem que pode ser conectado a repositórios no GitHub;
- Notifica o usuário (por e-mail) sobre resultados da build;
- Vasta documentação;
- Rápida curva de aprendizado.
