**SUMÁRIO**

[**3. Processos de Software** 2](#_Toc51250766)

[**3.1 Planejamento** 2](#_Toc51250767)

[**3.2 Processos de software: Incremental Adaptado** 2](#_Toc51250768)

[**3.2.1 Descrição das Etapas do Projeto** 3](#_Toc51250769)

[**4. Engenharia de Requisitos** 4](#_Toc51250770)

[4.1 Como descrever requisitos de usuário e requisitos de sistema 4](#_Toc51250771)

[4.2 Como definir as prioridades e riscos dos requisitos 4](#_Toc51250772)

[4.3 Processo de elicitação e análise de requisitos 4](#_Toc51250773)

[**5. Modelagem Conceitual (é necessário?)** 4](#_Toc51250774)

[5.1 Cenários ou Diagramas UML? 4](#_Toc51250775)

[**6. Projeto de Software** 5](#_Toc51250776)

[6.1 Arquitetura Game Engine (Core) 5](#_Toc51250777)

[6.2 Visão Geral de todas as ferramentas 5](#_Toc51250778)

[6.2.1 Compiladores 5](#_Toc51250779)

[6.3 Parser BPMN (Camunda) 6](#_Toc51250780)

[6.4 Método PYP – Play Your Process 6](#_Toc51250781)

[**7. Teste de Software** 6](#_Toc51250782)

[7.1 Plano de Testes 6](#_Toc51250783)

[7.2 Testes de Unidade Automatizadas 6](#_Toc51250784)

[7.3 Testes de Integração Automatizada 6](#_Toc51250785)

[7.4 Testes de Aceitação 6](#_Toc51250786)

[7.5 Documentação de Testes 6](#_Toc51250787)

[**8. Gerência de Configuração de Software** 7](#_Toc51250788)

[8.1 Slack – Comunicação entre a equipe 7](#_Toc51250789)

[8.2 Trello – Gestão de tarefas (Kanban) 7](#_Toc51250790)

[8.3 Git e Github – Controle de Versão 7](#_Toc51250791)

[8.3.1 Versionamento Semântico 7](#_Toc51250792)

[8.3.2 Workflow: Pull Request Workflow 7](#_Toc51250793)

[8.3.3 Workflow: No Switch Yard 7](#_Toc51250794)

[8.3.4 Guia de contribuição open source 7](#_Toc51250795)

[8.4 DevOps com Github, TravisCI e Codacy 7](#_Toc51250796)

[**10. Cronograma do Projeto** 7](#_Toc51250797)

[**11. Garantia de Qualidade** 8](#_Toc51250798)

# **3. Processos de Software**

Com o intuito de entregar resultados de alta qualidade e dentro do prazo estabelecido, através de um planejamento cuidadoso, foi estabelecido um escopo bem definido do projeto e as etapas necessárias para o desenvolvimento do motor de jogo. As etapas foram organizadas utilizando a metodologia de processo incremental e algumas práticas da metodologia ágil. A seguir, segue os detalhes do planejamento do projeto e como foram definidos os processos de software.

## **3.1 Planejamento**

O planejamento inicial é muito importante para definir bem o escopo do projeto. A ideia é que todos os envolvidos no projeto tenha uma visão clara da versão final do produto e dos processos de software para a execução do projeto. Na fase de planejamento deste projeto, foi definido as principais funcionalidades do software, assim como as limitações e os obstáculos a serem enfrentados durante o desenvolvimento.

Com a definição das principais funcionalidades, foi possível dividi-las em funcionalidades menores a um nível de visão suficiente para visualizar as dependências entre elas e definir a prioridade de cada uma. Por exemplo, para criar um sistema de renderização é necessário que o sistema de janelas esteja pronto. Portanto, a implementação do sistema de janelas tem a prioridade mais alta do que o sistema de renderização.

Com essa divisão de funcionalidades em requisitos mentores, foi possível definir as etapas iniciais de desenvolvimento (sprints) e a lista de requisitos com as prioridades e dependência para cada etapa.

## **3.2 Processos de software: Incremental Adaptado**

Existem diversas metodologias fundamentadas na engenharia de software e consolidadas no mercado atual que podem ser aplicadas em diferentes tipos de projeto de software. Ao mesmo tempo, a própria engenharia de software (e a experiência no mundo real) aconselha a adaptação de metodologias existentes para que o desenvolvimento do projeto consiga fluir da melhor maneira possível. Afinal, com o crescimento da complexidade dos softwares modernos e a possibilidade de uso em diversas áreas diferentes, é praticamente impossível definir uma metodologia que funcione completamente para todos os tipos de projetos.

Portanto, para este projeto foram adotados métodos e práticas de duas metodologias existentes: incremental e ágil.

A duração de cada etapa (sprint) de desenvolvimento é de quinze dias. Este tempo foi definido para coincidir com as datas de reuniões do projeto. A ideia é que sejam apresentadas nestas reuniões os resultados obtidos e a versão gerada do produto de cada etapa.

### **3.2.1 Descrição das Etapas do Projeto**

Cada etapa de desenvolvimento é dividida em diversas fases: a análise da sprint atual, definição da lista de requisitos e a versão desta etapa, desenvolvimento dos requisitos, testes de unidade, testes de integração e a geração da versão de software da sprint e atualização de todos os documentos.

#### 3.2.1.1 Análise da sprint atual

Nesta fase, é realizado a análise do estado atual do projeto (em geral, depois da apresentação dos resultados da reunião) e a definição de novos requisitos a serem desenvolvidos. Além disso, é definido também a versão esperada que deverá ser gerada no final da etapa. Seguindo os métodos do processo incremental, a ideia é que ao final de cada sprint, o usuário possa receber uma versão funcional do produto. Assim, ao final de cada etapa, o produto é incrementado com novas funcionalidades de versão em versão, até alcançar a versão final. Portanto, nesta fase, é muito importante executar os seguintes itens:

* Análise do estado atual do projeto
* Lista de novos requisitos para a sprint atual
* Definição da versão para a sprint atual (objetivo)
* Atualização dos documentos “Lista de Requisitos”

#### 3.2.1.2 Desenvolvimento

Esta fase é a fase de desenvolvimento dos novos requisitos definidos anteriormente. Todos os detalhes de desenvolvimento como convenção de código, fluxo de trabalho com o sistema de controle de versão e outras práticas relacionadas ao desenvolvimento estão descritas no documento “Guia de Desenvolvimento”.

**Nota:** Conforme o desenvolvimento, é possível que os documentos “Guia de Desenvolvimento” e “Lista de Requisitos” sejam atualizados constantemente.

#### 3.2.1.3 Teste de Unidade e Testes de Integração

A ideia principal de incluir testes de unidade no projeto é obviamente garantir a qualidade do produto, porém, a forma como foi implementada neste projeto, permite também agilizar certos processos de software. Utilizando conceitos de DevOps, foi possível empregar a automatização de testes e a Integração Contínua no projeto. A forma como os testes de unidade é implementada no projeto permite que ferramentas externas executem os testes de unidade de forma automática e verifique também se a nova implementação não afetou outros pontos do software. A ferramenta utilizada neste caso é o Travis CI. Todos os detalhes sobre este assunto estão descritos no capítulo mais adiante deste documento.

#### 3.2.1.4 Entrega da versão

Após o término de desenvolvimento de todos os requisitos, todos os documentos devem ser atualizados de acordo com os resultados obtidos e a nova versão do software deverá ser gerada com o número de versão correto. Com isto, basta apresentar a apresentação para a entrega da versão.

# **4. Engenharia de Requisitos**

## 4.1 Como descrever requisitos de usuário e requisitos de sistema

## 4.2 Como definir as prioridades e riscos dos requisitos

## 4.3 Processo de elicitação e análise de requisitos

* Descoberta e compreensão dos requisitos
* Classificação e organização dos requisitos
* Priorização e negociação dos requisitos
* Documentação dos requisitos

# **5. Modelagem Conceitual (é necessário?)**

## 5.1 Cenários ou Diagramas UML?

# **6. Projeto de Software**

Nesta seção abordamos todos os componentes que fazem parte do software, assim como todos os componentes utilizados para construí-lo. É descrita também a arquitetura de BPM Game Engine e o mapeamento utilizado no método Play Your Process.

## 6.1 Arquitetura Game Engine (Core)

## 6.2 Visão Geral de todas as ferramentas

### 6.2.1 Compiladores

Como a ideia do projeto é atender diversas plataformas, o projeto deve ser possível ser compilado por diversos compiladores. A seguir segue a lista de compiladores utilizados para compilar o projeto:

* GCC 9.2.0 (MinGW) – MinGW.org GCC Build-20200227-1

## 6.3 Parser BPMN (Camunda)

## 6.4 Método PYP – Play Your Process

# **7. Teste de Software**

## 7.1 Plano de Testes

## 7.2 Testes de Unidade Automatizadas

## 7.3 Testes de Integração Automatizada

## 7.4 Testes de Aceitação

## 7.5 Documentação de Testes

# **8. Gerência de Configuração de Software**

## 8.1 Slack – Comunicação entre a equipe

## 8.2 Trello – Gestão de tarefas (Kanban)

## 8.3 Git e Github – Controle de Versão

O sistema de controle utilizado neste projeto é o Git (<https://git-scm.com/>). A escolha de Git para o projeto é motivada pela sua popularidade e facilidade de integração com diversos projetos existentes. Além disso, ela é uma ferramenta livre e gratuita.

Para este projeto é recomendável utilizar qualquer versão a partir de **2.25.1**.



*Git Logo by Jason Long is licensed under the Creative Commons Attribution 3.0 Unported License.*

O projeto também com a plataforma online Github (<https://github.com/>) para a hospedagem do repositório do projeto. Esta plataforma também permite a integração com diversas ferramentas úteis e sistemas como o Travis-CI para automatização de builds e testes – práticas essências de integração contínua (DevOps).



## 8.3.1 Versionamento Semântico

## 8.3.2 Workflow: Pull Request Workflow

## 8.3.3 Workflow: No Switch Yard

## 8.3.4 Guia de contribuição open source

## 8.4 DevOps com Github, TravisCI e Codacy

# **10. Cronograma do Projeto**

# **11. Garantia de Qualidade**