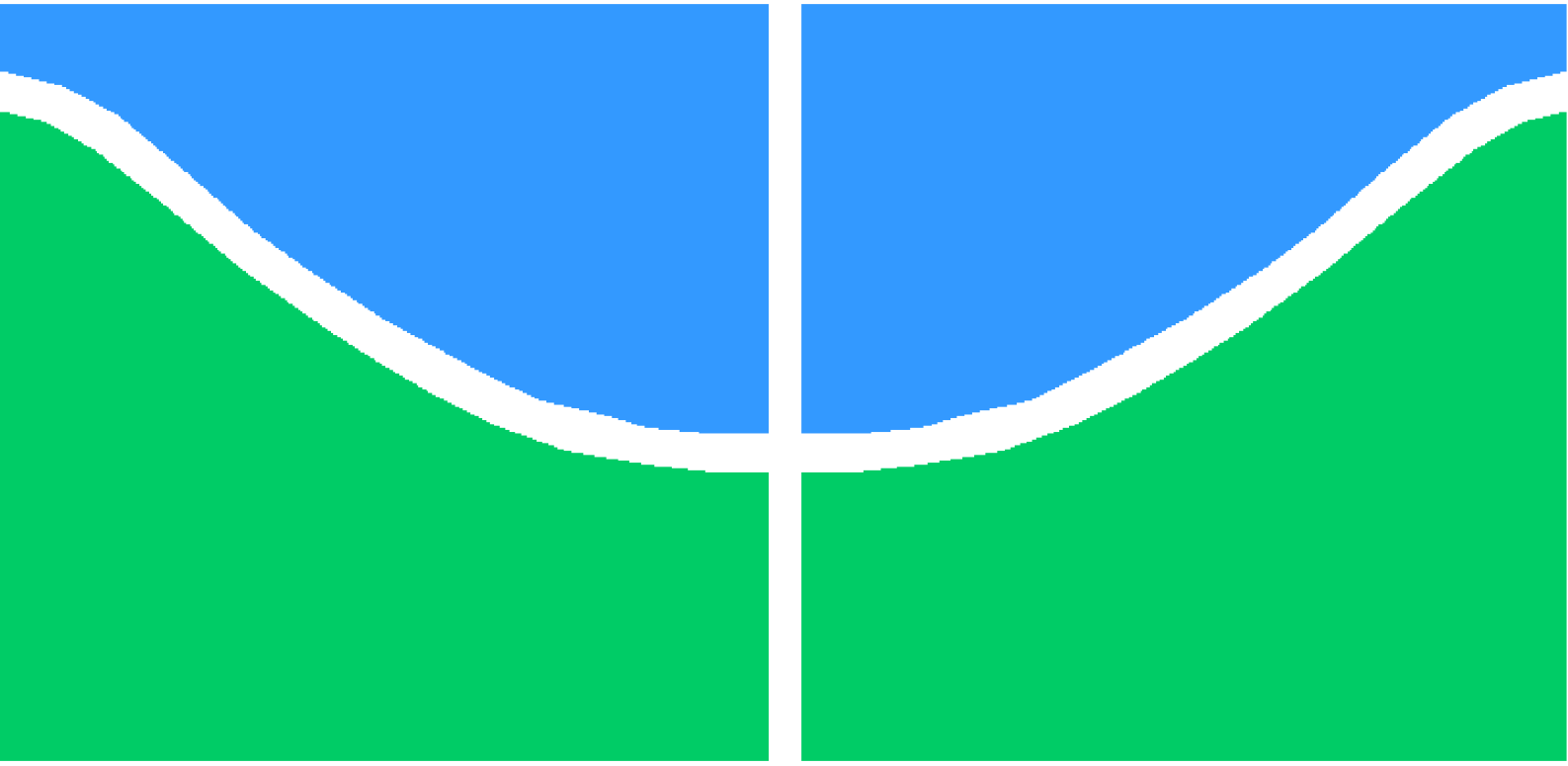


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Requisitos de Software - 201308

Relatório de Projeto

Grupo 01
Autores: Dylan, Eduardo, Wilton, Pedro
Orientador: George Marsicano Corrêa, MSc

Brasília, DF
2015

Dylan, Eduardo, Wilton, Pedro

Relatório de Projeto

Relatório referente à disciplina de Requisitos de Software, do curso de Engenharia de Software da Universidade de Brasília.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: George Marsicano Corrêa, MSc

Brasília, DF

2015

Dylan, Eduardo, Wilton, Pedro
Relatório de Projeto/ Dylan, Eduardo, Wilton, Pedro. – Brasília, DF, 2015-
Orientador: George Marsicano Corrêa, MSc

Relatório – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

I. George Marsicano Corrêa, MSc. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Relatório de Projeto

Brasília, DF
2015

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UP	Unified Process
RUP	Rational Unified Process
SAFe	Scaled Agile Framework
ER	Engenharia de Requisitos
PO	Product Owner
PM	Product Manager
IT	Investment Theme
EP	Epic
FE	Feature
US	User Story
TS	Task
FT	Functional Test

Sumário

1 Introdução

Este documento apresenta considerações gerais e preliminares relacionadas à redação de relatórios de Projeto de Graduação da Faculdade UnB Gama (FGA). São abordados os diferentes aspectos sobre a estrutura do trabalho, uso de programas de auxilio a edição, tiragem de cópias, encadernação, etc.

1.1 Visão Geral do Relatório

1.2 Seções do Relatório

2 Contexto

O Ministério das Comunicações é uma instituição pública que tem como áreas de competência os serviços de radiodifusão, postais e de telecomunicações, sendo responsável por formular e propor as políticas nacionais para estas áreas (MC, 2009). Possui a misão de desenvolver políticas públicas a fim de promover o acesso aos serviços de comunicações, contribuindo para o crescimento econômico, a inovação tecnológica e a inclusão social no Brasil (MC, 2009).

Dentro do ministério há a Coordenação Geral da Tecnologia da Informação (CGTI) responsável por atender as demandas internas de desenvolvimento de software. As demandas são requisitadas através de diversos sistemas de *software*. Sistemas esses que por serem utilizados no âmbito federal, tem um impacto direto para milhares de usuários, e indireto para milhões de cidadãos que necessitam que o órgao seja eficiente e eficaz em suas atividades, não prejudicando suas operações internas, e, consequentemente, externas.

Para tanto, a CGTI (mais precisamente a Divisão de Desenvolvimento de Sistemas - DSIS), necessita realizar a manutenção periódica dessas aplicações. Porém o proceso de manutenção de *software* é prejudicado pela dificuldade enfrentada tanto no gerenciamento quanto na execução das requisições que chegam à CGTI.

A grande quantidade de requisições contínuas tem evidenciado a falta de preparo da DSIS em conseguir atender essa demanda sem ter o seu fluxo de trabalho interrompido e prejudicado, sendo que, não raramente, algumas requisições acabam se perdendo no processo, pela incapacidade de acompanharem de forma adequada o processo.

A manutenção das aplicações ocorre após haver o processo de gerenciamento de mudanças. Cada requisição é atrelada à uma categoria, que define a prioridade do processo:

- Manutenção corretiva: eliminação de defeitos de códigos nos sistemas existentes;
- Manutenção adaptativa: adequações nas funcionalidades dos sistemas quando alteradas as regras e negócios, legislações e etc;
- Manutenção evolutiva: inserção de novas funcionalidades e/ou novos componentes nos sistemas existentes.

Há uma restrição quanto à quem pode requisitar as demandas, de acordo com a categoria atrelada ao processo:

- Manutenção corretiva: qualquer usuário dos sistemas utilizados no MC;

- Manutenção adaptativa e evolutiva: apenas o gestor do sistema.

Dentro dessas categorias, o órgão necessita mapear o fluxo de manutenção, realizando:

- A priorização das requisições;
- A execução das manutenções;
- O gerenciamento das execuções;
- A enrega ao cliente solicitante.

Sendo assim, o grupo 1, observando o contexto apresentado, deve apresentar uma solução que melhore o gerenciamento de requisições de manutenção de sistemas, dentro do Ministério das Comunicações.

3 Escolha da Abordagem

No desenvolvimento de um software é imprescindível a utilização de alguma abordagem que conduza seu processo produtivo.

Uma abordagem se refere a uma metodologia que será usada para estruturar, planejar, e controlar o processo de desenvolvimento do sistema (CMS, 2005).

Logo, neste capítulo será justificada a escolha de uma determinada abordagem feita pelo grupo 1 da disciplina Requisitos de Software, responsável por realizar a manutenção de sistemas de software mantidos pelo Ministério das Comunicações. Primeiro será exposto uma visão geral de cada abordagem, explicando suas diferenças e os pontos positivos e negativos presentes em cada uma delas, para que enfim uma justificativa seja apresentada. As abordagens escolhidas para representar cada uma das correntes, tradicional e adaptativa, são o Rational Unified Process (RUP) e o Scaled Agile Framework (SAFe), respectivamente.

3.1 Abordagem Tradicional - RUP

O Processo Unificado surgiu em meados dos anos 90 e tem como proposta reunir diversas práticas e filosofias que se provaram eficientes até então no contexto de desenvolvimento de software (KRUCHTEN, 2003, p. 45).

Valoriza conceitos como arquitetura de software, desenvolvimento iterativo, gerência de requisitos, construção de modelos visuais para descrever o sistema, entre outros (KRUCHTEN, 2003, p. 45). A abordagem é baseada em disciplinas, onde cada disciplina agrupa uma série de atividades relacionadas e cada atividade produz diversos artefatos, que por sua vez contêm informações do processo e são submetidos à controle de versão (são produzidos, modificados e evoluídos durante o ciclo de vida do projeto). As disciplinas representam áreas de interesse presentes no processo de desenvolvimento de software (KRUCHTEN, 2003, p. 45).

O RUP estabelece a divisão do processo em quatro fases, sendo que cada fase possui um marco (objetivo principal) que será almejado durante sua execução. As fases por sua vez são divididas em iterações que consistem em um conjunto de atividades a serem realizadas para se obter um incremento do produto. Argumenta que o planejamento a longo prazo do projeto fornece uma visão à equipe de desenvolvimento do que deve ser feito e quando deve ser feito para que o projeto seja concluído e entregue no prazo. Analisando o processo unificado sob a perspectiva da disciplina de requisitos, é possível observar que o principal recurso para se representar requisitos dentro do processo são

os casos de uso (KRUCHTEN, 2003, p. 45). Um caso de uso descreve uma interação entre uma entidade e o sistema e demonstra uma capacidade do software, assim como as funcionalidades que serão oferecidas ao usuário (KRUCHTEN, 2003, p. 124-125). Para atingir os objetivos da ER no RUP, a disciplina de requisitos descreve como definir uma visão do sistema, traduzir essa visão em um modelo de caso de uso (que, com especificações suplementares, definirá os requisitos detalhados do software), e como usar os atributos dos requisitos para ajudar a gerenciar o escopo do projeto e como mudar os requisitos do sistema (KRUCHTEN, 2003, p. 182-183).

Por fim, o RUP é um framework implementado por grandes empresas, mas determina que sua customização é aceita para que ele se adeque à realidade de organizações de menor porte, sendo possível definir as atividades e práticas do RUP que sejam do interesse da empresa em questão, desde que não sejam feridos os seus principais valores. As grandes empresas que o utilizam não o utilizam sempre da mesma maneira: enquanto algumas o utilizam de maneira formal, outras o customizam completamente (KRUCHTEN, 2003, p. 57).

3.2 Abordagem Adaptativa - SAFe

Bem mais novo, o Scaled Agile Framework (SAFe) traz ideias novas, mas tambem aspectos já consolidados na área de metodologias de software (Scrum, Kanban, etc).

Adaptável em diversos pontos, traz a ideia de dividir o projeto em três níveis: Portfolio, Program e Team. Cada nível produz seus artefatos, dispõe de diferentes papeis, tem responsabilidades, e interagem de maneira diferente (LEFFINGWELL, 2010, p. 124-125).

Uma das características mais importantes em relação ao UP, é que, enquanto o UP é dirigido a UC, possuindo assim somente ele como descritor de comportamentos desejáveis para o sistema, no SAFe, existem três níveis de abstração para descrever tais comportamentos: o Epic, a Feature e a User Story (LEFFINGWELL, 2010, p. 182-183). O Epic é bem mais alto nível, a Feature mais baixo nível que o Epic, e a User Story mais baixo nível que ambos (LEFFINGWELL, 2010, p. 182-183).

3.3 Escolha da Abordagem e Justificativa

Considerando os conceitos levantados à respeito das duas abordagens, é possível inferir que ambas fornecem diversas práticas e filosofias que podem agregar valor ao nosso processo e podem nos conduzir ao nosso objetivo: realizar a manutenção de sistemas para o Ministério das Comunicações. Desde os primórdios do desenvolvimento de software, uma das principais causas responsável pelo fracasso de projetos é a utilização de uma

abordagem linear proveniente das engenharias tradicionais, tendo em vista que no contexto da engenharia de software na maioria das vezes não é possível determinar completamente os requisitos de um sistema que será desenvolvido, o que inviabiliza a implantação de um processo sequencial.

Ambas metodologias citadas neste documento como exemplo procuram contornar este problema através da implementação de diversos conceitos que são comprovadamente eficientes no desenvolvimento de software, tais como gerência de requisitos e desenvolvimento iterativo e incremental. As propostas de ambas metodologias são embasadas em uma série de argumentos consistentes e temos certeza que apesar de suas diferenças sob o ponto de vista da adaptabilidade e previsibilidade, ambas propõe a aplicação de diversas das melhores práticas existentes no contexto de processos de desenvolvimento de software, o que nos leva a concluir que independente da abordagem escolhida, estaríamos bem equipados para desenvolver o projeto.

Finalmente, a equipe optou pela abordagem adaptativa pelos seguintes pontos:

- Todos os integrantes apreciam e valorizam os aspectos levantados pelo manifesto ágil
- O desenvolvimento de software iterativo (no contexto adaptativo, baseado em sprints) nos permitirá entregar incrementos constantes do sistema, permitindo demonstrar a evolução contínua do software ao cliente
- A representação dos requisitos do sistema em diferentes níveis de abstração (features e estórias) nos auxiliará a compreender as necessidades do cliente, tendo em vista que é possível avaliar os requisitos tanto sobre um ponto de vista macroscópico, onde é possível avaliar as principais capacidades do produto, quanto sobre um ponto de vista microscópico, onde é possível definir requisitos que podem ser entregues em uma única iteração
- A divisão interna do nível de time é considerada interessante pela equipe, levando em conta a importância de um ScrumMaster em qualquer organização e os benefícios trazidos pela proximidade entre uma pessoa que tenha autoridade para representar os clientes (Product Owner) e a equipe de desenvolvimento. A equipe considera essa relação e interação crucial para a execução do processo.

Apesar da experiência da equipe em projetos que utilizam uma abordagem adaptativa, ela admite que os principais frameworks de metodologias ágeis são aplicáveis à equipes de pequeno e médio porte. A promessa do SAFe de estabelecer uma abordagem adaptativa para organizações de grande porte estimulou nossa curiosidade e levando em consideração o primeiro ponto apresentado, nós valorizamos a coragem para implementar um framework (SAFe) nunca antes utilizado por nenhum integrante da equipe.

É importante ressaltar que valorizamos ambas as abordagens e reconhecemos os benefícios trazidos por elas. Portanto, procuramos enaltecer os pontos positivos da abordagem escolhida, e ressaltamos que os pontos levantados não são uma resposta direta à abordagem tradicional (não é porque um ponto está presente na abordagem adaptativa que ele não é proposto pela abordagem tradicional). Preferimos construir nossa justificativa em cima dos aspectos considerados beneficentes à nós, no lugar de construí-la em cima dos aspectos negativos da abordagem tradicional.

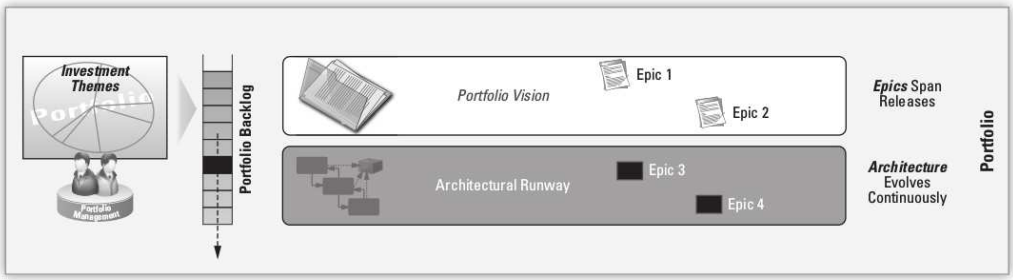
4 Engenharia de Requisitos

Na seção a seguir será dada uma pequena introdução dos três níveis de projeto do SAFe (Portfolio, Program, Team), e será dito um pouco sobre esses níveis no contexto de projeto do grupo 1. Certas adaptações foram feitas com base nas características dos integrantes de ambos os grupos (Requisitos e MPR). Então, na seção seguinte (*Papéis*) será falado sobre os papeis principais do SAFe no contexto do grupo. Certos papeis foram retirados (por serem julgados como desnecessários para o contexto do projeto), e, portanto, não serão citados nem explicados. Papeis utilizados serão explicados, e, caso eles sejam alterados para que funcionem de maneira diferente à referida na bibliografia, essas alterações serão explicadas e justificadas.

4.1 Scaled Agile Framework

4.1.1 Nível de Portfólio

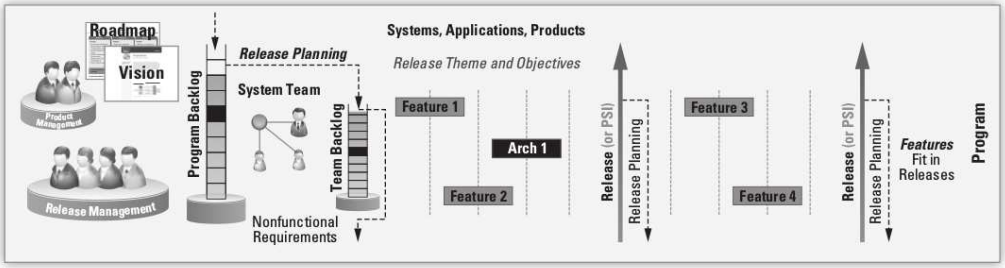
Figura 1: Nível de Portfólio no SAFe - (LEFFINGWELL, 2010, p. 44)



Responsável pelos artefatos *epics* e *investiment themes*, este nível de projeto traz os requisitos com maior nível de abstração dentre os três níveis. Contém um tipo de time (*portfolio management team*), costuma ter seu próprio backlog (*portfolio backlog*), e apresenta os conceitos de *portfolio vision* e de *architectural runway* (LEFFINGWELL, 2010, p. 227-228). No contexto do grupo 1 esse nível de projeto será bastante customizado, pois julga-se o tamanho do projeto como pequeno, fazendo-se desnecessária a utilização de certos papéis e certos conceitos (costumam ser utilizados para grandes projetos, e a não utilização de tais não prejudicará a qualidade do processo). Ao invés de se usar o *portfolio backlog*, os épicos entre outros insumos nascidos nesse nível serão armazenados no *program backlog*. Essa escolha foi feita pois o épico ocorre em um nível muito macro para o nível do projeto, onde o grupo voltaria a este possível *backlog* pouquíssimas vezes durante o projeto. Logo preferiu uni-lo ao *backlog* de Nível de Programa, deixando um papel cuidando de ambos.

4.1.2 Nível de Programa

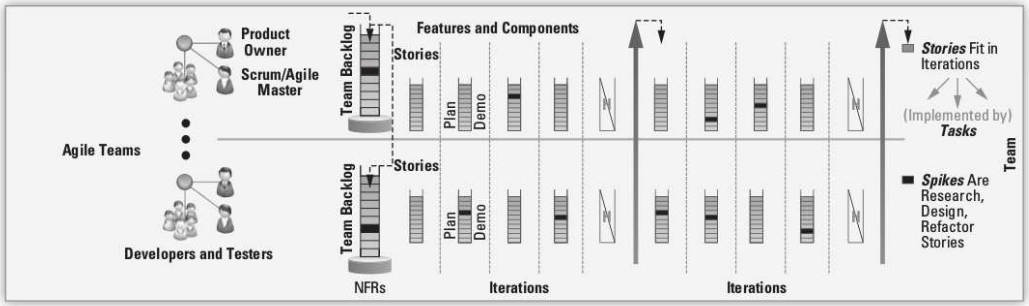
Figura 2: Nível de Programa no SAFe - (LEFFINGWELL, 2010, p. 39)



Esse nível tem como principais objetivos manter a visão (Documento de Visão), gerenciar a *release*, gerenciar a qualidade (integrando os resultados obtidos pelos times, e garantindo que os padrões de qualidade, performance, entre outros, estão sendo assegurados), fazer o *deploy* do sistema, gerenciar recursos (ajustando prazo e gastos), entre outras tarefas (LEFFINGWELL, 2010, p. 63-64). No contexto do grupo 1, os integrantes de MPR fornecerão dados para a escrita das *features* (principalmente sob o papel de PO, já que não será utilizado o papel de PM), os alunos de Requisitos escreverão as features e as documentarão no Documento de Visão. Como já dito, haverá o *program backlog*, onde serão armazenadas *features*, épicos, entre outras coisas.

4.1.3 Nível de Time

Figura 3: Nível de Time no SAFe - (LEFFINGWELL, 2010, p. 34)



A unidade básica de trabalho para este nível é a *estória de usuário*. O objetivo deste nível é definir, construir e testar as estórias de usuário no escopo de uma *sprint*, afim de se concluir mais partes do produto final (LEFFINGWELL, 2010, p. 47-48). No contexto do grupo 1, os alunos de MPR terão o papel de *product owner*, fornecendo os insumos necessários para a escrita das estórias, que serão escritas em conjunto entre PO e desenvolvedores. Além disso, os alunos de Requisitos farão todo o processo de alimentação do *backlog* de time, processo de planejamento da *sprint* (serão os *Scrum*

Masters), priorização das estórias com base no WSJF, desenvolverão e testarão as soluções (com base nos critérios de aceitação).

4.2 Papéis

Um papel define comportamentos e responsabilidades de um indivíduo ou de um grupo de indivíduos que trabalham juntos como um time (KRUCHTEN, 2003, p. 61-65). O corportamento é expresso em termos de atividades que o papel pratica, e, cada papel é associado a um conjunto de atividades. No SAFe existem diferentes papéis para os diferentes níveis do sistema. No Nível de Portfólio vão haver, principalmente, papéis que irão interagir com os épicos e temas de investimento, no Nível de Programa, papéis que irão interagir com as *features*, e, no Nível de Time, papéis que irão interagir com estórias de usuário.

Nesta seção será dada uma breve definição de papeis que serão utilizados, justificativas e explicações para papeis customizados, e quem representa este papel no contexto do grupo 1.

4.2.1 Papéis no Nível de Portfólio

4.2.1.1 Epic Owner

Responsável por definir e analisar o trabalho que será seguido no épico (LEFFINGWELL, 2010, p. 418-419). No contexto do grupo 1, afim de não descaracterizar a estrutura do SAFe, este papel terá o papel de gerenciar o Nível de Portfólio e de ser o responsável pelos épicos.

4.2.2 Papéis no Nível de Programa

4.2.2.1 Product Manager

Responsável por entender as necessidades do cliente, documentar, priorizar e validar requisitos (no nível de *feature*), gerenciar mudanças, entre outras coisas (LEFFINGWELL, 2010, p. 283-287). No contexto do grupo 1, o Gerente do Produto (*product manager*, PM) será fortemente customizado. Pois uma de suas principais atribuições é gerencias os diversos PO's, quando existem. Como no contexto do grupo haverá apenas um PO. As tarefas do PM serão atribuidas ao PO.

4.2.2.2 Release Management

Consistindo do Product Manager e do Release Train Engineer, o papel de Release Management é responsavel por comunicar o *status* da release aos *stakeholders*, coordenar

com a gerência do produto e gerência de *marketing* as comunicações internas e externas, validar a qualidade relevante do produto de acordo com critérios, prover a autorização final da *release*, ajudar a adaptar e inspecionar o processo de *release*, entre outras coisas (SCALED AGILE INC., 2015). No contexto do grupo 1, o Release Management será feito somente por quem atua no papel de *Release Train Engineer*, no caso, os alunos de Requisitos. O papel não será compartilhado com os alunos de MPR pois, como já dito, pratica atividades com mais relação com a disciplina de Requisitos. Será um papel flutuante, onde cada semana um aluno do grupo terá o papel de ser /emphrelease manager.

4.2.2.3 Business Owners

Pequeno grupo de *stakeholders* de grande confiança, que dispõe de capacidade de administração, responsáveis pela gestão, qualidade, implantação e desenvolvimento de software (SCALED AGILE INC., 2015). No contexto do projeto, diferente da bibliografia, eles não serão responsáveis pelo desenvolvimento e implantação do *software*, e será praticado pelos alunos de MPR.

4.2.3 Papéis no Nível de Time

4.2.3.1 Product Owner

Responsável por representar o interesse de todos os envolvidos no projeto, faz parte do time e esta junto no dia-a-dia dos desenvolvedores e testadores, elaborando as estórias de usuário e ajudando o time a alcançar seus objetivos (LEFFINGWELL, 2010, p. 47-48) Além de ser um membro que possui visão do modelo de negócio. No contexto do grupo 1, o *product owner* não irá escrever as estórias. O papel de PO será delegado a um integrante de MPR que deverá validar e analisar a escrita das estórias que será feita pelos integrantes do grupo de Requisitos.

4.2.3.2 Scrum Master

É responsável por facilitar o progresso do time (ajudando assim a se alcançar o objetivo final), liderar os esforços do time, reforçar as regras do processo ágil e eliminar impedimentos (LEFFINGWELL, 2010, p. 47-48). No contexto do grupo 1, o papel de *Scrum Master* irá flutuar durante o projeto, e será escolhido um integrante do grupo de Requisitos para ser *Scrum Master* a cada *sprint*. Essa rotatividade irá fazer com que todos os integrantes aprendam a ser um *Scrum Master*, e, só é necessário um por *sprint*.

4.2.3.3 Desenvolvedores e Testadores

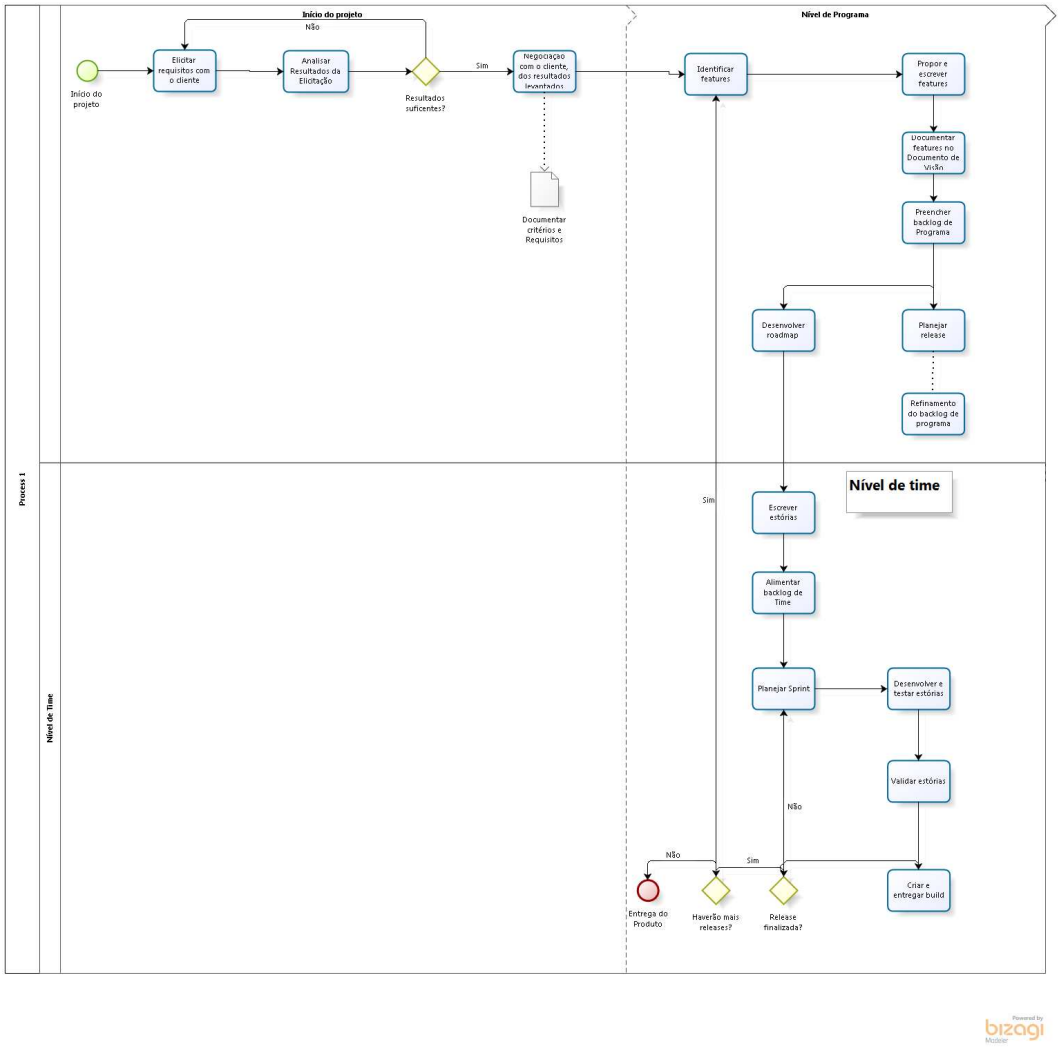
Responsáveis por desenvolverem e testarem as estórias de usuário. No contexto do grupo 1, todos os integrantes de Requisitos farão parte desse papel, e também será

responsável por escrever as estórias de usuário e os critérios de aceitação. Este papel ainda será responsável pelo desenvolvimento da solução final e por testar as estórias.

4.3 Atividades

Serão detalhadas nessa seção o conjunto de atividades a serem feitas durante o projeto, e quem está envolvido na atividade. Modelagem geral de atividades:

Figura 4: Modelagem das atividades, feita no *Bizagi*



Lista de atividades:

- Elicitar requisitos com o cliente: no contexto do grupo 1, atividade feita pelo *epic owner* (Requisitos) e cliente (MPR), consiste em entender de maneira macro o contexto e as necessidades do projeto.
- Analisar resultados da elicitação: Atividade feita pelos *epic owners*, consiste em se analisar os insumos fornecidos pelo cliente na atividade anterior, buscando-se

soluções plausíveis e pontos de vista diferentes.

- Negociação com o cliente dos resultados levantados: feita entre *epic owner* e cliente, consiste em expor os pontos de vista descobertos na análise, tentando encontrar um ponto comum entre a visão do cliente e da equipe.

- Documentar critérios e requisitos: documentar de maneira formal as necessidades e descrições necessárias no projeto conseguidos na negociação, tudo em um nível bem macro. Feito pelo *epic owner*, guiará a visão do épico.

- Identificar *features*: consiste em identificar as possíveis *features* do sistema. Feita em conjunto entre o *Product Owner* (que, como já dito, no contexto do grupo 1 terá as tarefas do PM) e o RTE (alunos de Requisitos), será salva no *backlog* de programa.

- Propor e escrever features: com as possíveis *features* identificadas, se chega a um ponto comum se tal *feature* deve estar presente ou não. Se sim, serão escritas e salvas. Feito em conjunto entre o PO e o RTE.

- Documentar *features* no Documento de Visão: Feito pelo RTE (alunos de Requisitos), consiste em documentar as *features* levantadas no Documento de Visão.

- Preencher *backlog* de programa: consiste em documentar o conjunto de *features* disponíveis no *backlog* de programa, dando assim uma melhor visão para o time de qual solução pode ser implementada em determinado momento. Feita pelo Release Manager.

- Planejar *release*: consiste em definir quais *features* serão implementadas na *release*, períodos, visões, etc. Feita em conjunto entre os papéis de Release Manager, PO e RTE.

- Refinamento do *backlog* de programa: Consiste principalmente em dizer o estado atual de cada *feature* que deve ser implementada na *release* atual. Tarefa feita pelo Release Manager.

- Escrever estórias de usuário: consiste em documentar as estórias de usuário advindas de uma determinada *feature*. Feita em conjunto entre PO, Desenvolvedores e Scrum Master.

- Alimentar *backlog* de time: consiste em documentar e armazenar o conjunto de estórias no backlog, garantindo também sua rastreabilidade. Feito pelo Scrum Master.

- Planejar *sprint*: Feito em conjunto entre PO, Desenvolvedores e Scrum Master, consiste em planejar o que será implementado na *sprint* seguinte, quem fica com determinada tarefa, pontos falhos da *sprint* passada, etc.

- Desenvolver e testar estórias: tarefa feita pelos desenvolvedores e testadores, consiste em desenvolver determinada estória de usuário, testar e validar com base nos critérios de aceitação.

- Criar e entregar *build*: Feita pelos desenvolvedores, consiste em empacotar o desenvolvimento das estórias feitas numa determinada sprint, transformar em produto, e entregar este incremento ao usuário.

5 Técnicas de Elicitação de Requisitos

Descobrir requisitos é um dos grandes desafios do processo de produção de Software. Não ser capaz de completar essa tarefa da maneira certa e com qualidade impossibilitará o projeto de obter sucesso, não importa quão bom sejam as qualidades do time (LEFFINGWELL, 2010, p. 227-228).

Por tal motivo, existe a necessidade de utilização de técnicas que ajudarão a levantar requisitos. Cada técnica apresenta melhor resultado em determinados contextos, e, muitas vezes, elas se completam. Por tal motivo, as técnicas foram escolhidas baseadas nas características do time.

As características que mais influenciaram na escolha das técnicas foram:

- Disponibilidade: técnicas que exigem grande quantidade de encontros são mais difíceis de serem aplicadas pois os horários em que os times poderiam aplicá-las não é suficiente, embora seja flexível.
- Experiência: certas técnicas exigem um certo nível de experiência para apresentar resultados satisfatórios. O grupo não tem experiência com as técnicas ou as metodologias, logo, opta-se por técnicas mais fáceis.

5.1 Workshop de Requisitos

5.2 Brainstorming

5.3 Entrevistas e Questionários

5.4 Mock-Ups

5.5 Product Council

5.6 Análise Competitiva

5.7 Customer change request system

5.8 Modelagem caso-de-uso

6 Tópicos de Gerenciamento de Requisitos

6.1 Rastreabilidade de Requisitos

A Rastreabilidade de Requisitos (descrever, mostrar e acompanhar a vida de um requisito nos sentidos *forward-to* e *forward-from* (GARCIA, 2014)) é tido como uma das soluções para o problema da dessincronização entre *software* e seus requisitos, sendo que a solução desse problema é quase que uma exigência básica da industria de desenvolvimento de *software* atual (JR, 2007). Faz parte de uma *Specific Practice* da área de processo *Manage Requirements* do CMMI. É divida entre *forward-to* (itens que derivam de um requisito x) e *forward-from* (itens que são derivados de um requisito x).

Evidentemente que a Rastreabilidade de Requisitos muda de abordagem para abordagem. Será mostrada então o formato de rastreabilidade do grupo 1, levando em consideração que o framework de abordagem utilizado é o SAFe.

Primeiro, é lembrado que do tema de investimento será derivado o épico. Um épico pode ser classificado em arquitetural ou de negócio, e dele derivarão *features*. Das *features* serão derivadas estórias de usuário, que delas (as estórias de usuário) derivam tarefas e, no nosso contexto, testes funcionais.

A nomenclatura utilizada será uma concatenação entre prefixo (mostrados abaixos) e sufixos (número do índice do item). Lista de prefixos utilizados:

Item	Prefixo	Prefixo+sufixo (exemplo)
Tema de Investimento	IT	IT0004
Epico	EP	EP0002
Feature	FE	FE0033
Estória de Usuário	US	US0243
Tarefas	TS	TS0932
Teste Funcional	FT	FT2142

6.2 Atributos de Requisitos

Requisitos, assim como objetos do mundo real, possuem atributos que os descrevem. Um carro, por exemplo, tem determinada potência, consome determinada quantidade de combustível, etc. Os requisitos são da mesma forma: apresentam atributos que os descrevem (IBM RATIONAL DOORS, 2010).

Existem diversas maneiras de se analisar qual requisito deve receber maior prioridade, e isso quase sempre é feito levando em consideração seus atributos. No contexto

deste projeto, será utilizado o *High Delay Cost First* em conjunto com o *WSJB - Weighted Shortest Job First*, também sugerido por (LEFFINGWELL, 2010). O *High Delay Cost First* consiste em se dar prioridade as *features* que apresentam maior penalização a cada atraso, enquanto o *WSJB* consiste em se dar prioridade as *features* que necessitam de menos esforço, mas que punem maior a cada atraso. Os dois se complementam, e no final das contas se faz a feature com o maior *weight* (peso), que é dado pela fórmula *Peso = Custo do atraso / Esforço* (LEFFINGWELL, 2010, p. 266).

Sendo assim, devemos ter como atributos de requisitos pelo menos os necessários para utilizar o HDCF e o WSJF. Eles são: *esforço, dificuldade, custo, tempo, prioridade.* Mas, em conjunto, também utilizaremos alguns atributos sugeridos por (IBM RATIONAL DOORS, 2010).

Sendo assim, os atributos de requisitos utilizados pelo grupo 1 serão:

- Fonte: De onde este requisito é derivado? Pessoa, documento, outro requisito, etc.
- Prioridade: Este requisito tem prioridade alta, média ou baixa?
- Comentários: Comentários de quem documentou tal requisito.
- Status: Condição atual do requisito (concluído, não começado, ou em progresso).
- Risco: Qual o risco de determinado requisito durante sua implantação? Podendo ser alto, médio ou baixo.
- Prazo: Até quando tal requisito deve estar pronto?
- Nível de teste: Qual o nível de teste de determinado requisito? Poderá ser nulo, médio, ou suficiente.
- Esforço: Quanto esforço é necessário para se concluir determinado requisito? Poderá ser alto, médio ou baixo.
- Custo do atraso: Quão grande é a punição por atraso de determinado requisito? Poderá ser alto, médio ou baixo.

Na fórmula citada acima (WSJB), será assumido valor 3 para alto, 2 para médio e 1 para baixo para os atributos correspodentes. Exemplo: Um atributo com custo do atraso alto e esforço médio:

$$Peso = C/E = 3/2 = 1.5 \tag{6.1}$$

Onde: C = Custo do atraso, E = Esforço.

Contra um requisito com custo do atraso baixo e esforço alto:

$$Peso = 1/3 = 0.3333 \tag{6.2}$$

Nesse caso, o primeiro requisito seria implementado primeiro, pois apresenta maior peso.

7 Ferramentas de Gerência de Requisitos

"If the process to select a RM tool seems daunting, the process to implement a tool is an even greater challenge"(BEATTY, 2013).

É mandatório a utilização de uma ferramenta de RM para times que desejam boa organização e qualidade no processo de desenvolvimento. Contudo, essa tarefa é difícil, e como dito na citação acima, mais difícil ainda é convencer um time a utilizá-la. Por tais motivos, pesquisou-se não somente por ferramentas completas: também analisou-se critérios como curva de aprendizagem, adaptação, customização, entre outros.

Assim sendo, foram inspecionadas mais a fundo três ferramentas de gerenciamento de requisitos: *Rally*, *Jama* e *Polarion*. Devido a importância que o grupo dá a realização de um trabalho colaborativo e participativo, foi fator imprescindível a ferramenta ser *web-based*, facilitando o acesso independente da plataforma de trabalho, ou lugar em que a equipe se encontre.

Embora a utilização de uma ferramenta instalável em um servidor pudesse ser uma opção alternativa que forneceria os mesmos benefícios descritos acima, decidiu-se que o esforço e tempo demandado para a instalação e configuração não traria benefícios suficientes, ainda mais considerando o contexto do trabalho.

Outro fator fundamental foi a compatibilidade com a metodologia adotada pela equipe.

Sendo assim, as ferramentas testadas serão apresentadas uma a uma, contemplando suas características, e, ao final da apresentação das ferramentas, será mostrada a decisão final e a justificativa.

7.1 Ferramenta 1 - Jama

- Suporte a múltiplos projetos
- Suporte a múltiplas equipes e papéis
- Importação de dados (doc/docx, xls/xlsx, xml, csv, arquivo de exportação do IBM Rational Doors)
- Impressão e exportação dos dados (doc, xls)
- Suporte nativo a: releases, sprints, épicos, estórias de usuário, pontos da estória, cenários de uso, e casos de teste

- Rastreabilidade dos requisitos
- Integração com outros serviços

O Jama possui uma interface agradável, de baixa curva de aprendizagem, e com boas opções de gerenciamento de requisitos voltado para a abordagem adaptativa.

O padrão de criação de requisitos é baseado em campos de formulários e em modelos pré-definidos de documentos de texto que permitem uma flexibilização muito grande em sua criação, sendo bastante útil quando se quer seguir um padrão próprio ou alternativo na criação dos requisitos.

A ferramenta ainda apresenta um boa rastreabilidade dos requisitos, relacionando os épicos com as estórias de usuários e com os casos de testes, funcionando em ambos os sentidos (ida e volta).

A falta de alguns componentes gráficos como *kanban* e *backlog*, que seriam interessantes no contexto adaptativo fazem falta, mas podem ser facilmente substituídos por ferramentas que se integram ao Jama.

A ferramenta apresenta algumas informações gerais sobre o projeto, dentre eles: risco da iteração e gráfico de priorização de estórias. E uma das grandes vantagens da ferramenta é o eficiente gerenciador de revisão, que permite comparar as edições realizadas pelos usuários nas estórias, épicos, e etc, permitindo a restauração dos documentos para uma versão anterior.

7.2 Ferramenta 2 - Rally

- Suporte à múltiplos projetos
- Suporte à múltiplas equipes e papéis
- Interface prática, amigável e organizada
- Rastreabilidade dos requisitos
- Diversas opções de estatísticas em forma de gráfico sobre o projeto
- Suporte nativo a: *releases*, epicos, funcionalidades, estórias de usuário, iteração, pontos da estória, casos de teste, *kanban* e *backlog*

O Rally se destaca por sua interface funcional e prática, sendo bastante informativo e organizada. Pelo fato de possuir um kanban com cinco estágios, diversas opções gráficas para exibir o progresso do projeto, além de muitas informações relevantes para

o gerenciamento do projeto e requisitos, a ferramenta se mostra poderosa e focada em projetos que usem abordagens adaptativas.

A rastreabilidade funciona muito bem quando se vai de épico para estórias de usuários, mas o caminho contrário não é possível, pois há uma ligação apenas para os componentes filhos. No entanto isso pode ser contornado usando algumas das funções de visualização gráfica disponíveis na ferramenta.

Há apenas a opção de exportar os dados para arquivos PDFs, mas de forma individual, o que não se mostra muito útil. A possibilidade de se inserir arquivos anexos e a liberdade de criar estórias de usuário utilizando um editor de texto avançado permite uma flexibilização interessante das descrições das estórias.

Embora o gerenciador de revisão da ferramenta seja de fácil acesso, ela não tem funções importantes como comparação entre diferentes versões de um arquivo, e muito menos a opção de se restaurar o arquivo para uma versão anterior, o que o torna um tanto inútil, servindo apenas como log de alterações.

7.3 Ferramenta 3 - Polarion

- Suporte a múltiplos projetos;
- Suporte a múltiplas equipes e papéis;
- Importação de dados (doc/docx e xls/xlsx)
- Exportação de dados (doc)
- Interface simples
- Rastreabilidade dos requisitos
- Algumas opções de estatistícas em forma gráfica sobre o projeto.

O Polarion se assemelha muito com um repositório de documentos editáveis, o que possui vantagens e desvantagens.

Como desvantagens, o sistema se distancia bastante de um sistema elegante e moderno de gerenciamento de requisitos, e não possui um suporte direto a metodologias ágeis, já que há nativamente apenas requisitos funcionais, requisitos do usuário, casos de testes e *features*.

A vantagem desse sistema é que tudo funciona em forma de *templates*, ou seja, é possível personalizar os itens para assumirem a forma de épicos, mas tudo acontece como se estivesse produzindo um documento de texto. Mas não é por isso que a ferramenta

deixa de apresentar recursos importantes como status, prioridade, rastreabilidade com ligações entre requisitos de forma bidirecional e histórico de versão, inclusive com opções de comparação entre as revisões, mas sem uma opção específica para a restauração do documento para uma versão anterior.

O Polarion é uma ferramenta versátil, que pode coibir os desavisados que ainda temem editar requisitos em forma de documentos de texto, mas basta dar uma olhada mais aprofundada para perceber que a ferramenta cumpre com os requisitos que qualquer gerenciador de requisitos deva possuir.

7.4 Ferramenta Escolhida e Justificativa

De todas as ferramentas testadas era esperado que o Rally fornecesse os componentes necessários para sua adoção como ferramenta de gerenciamento de requisitos, devido a sua forte integração com a abordagem adaptativa, no entanto, dois fatos considerados de grande importância acabaram tirando a ferramenta, inicialmente preferida, da disputa: a impossibilidade de rastrear bidirecionalmente os requisitos de forma fácil, e também a incapacidade de comparar e restaurar as modificações realizadas nos requisitos.

A Polarion mostrou-se muito eficiente nas funções fundamentais de gerenciamento de requisitos, surpassando as expectativas e até mesmo mostrando-se mais indispensável que a ferramenta Rally. No entanto, por ser pouco adaptada aos métodos ágeis, ter uma forma de gerenciamento de requisitos considerada arcaica, e principalmente por não ter características interessantes e úteis presentes nas outras ferramentas, também optou-se por não utilizá-la.

Sendo assim, optou-se pela ferramenta Jama. Poderia-se dizer que foi a escolha que sobrou, mas isso não faria jus ao que a ferramenta demonstrou. Sem dúvida, é a ferramenta mais completa entre as testadas, e embora não tenha uma interface tão agradável ou funcionalidades tão chamativas quanto as que são apresentadas na ferramenta Rally, possui o que é essencial, no que as demais ferramentas sempre deveriam fornecer: fácil e sólida rastreabilidade. A Jama, além de tudo é muito preparada para o contexto ágil e possui integração nativa com ferramentas que extendem suas funcionalidades para outros contextos dentro do projeto de software, não deixando a equipe presa à ferramenta para gerência de outras coisas.

Referências

BEATTY, J. *Winning the Hidden Battle: Requirements Tool Selection and Adoption*. Austin, TX USA, 2013. Citado na página 35.

CENTERS FOR MEDICARE & MEDICAID SERVICES. *Selecting a Development Approach*. [S.l.], 2005. Disponível em: <http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf>. Citado na página 17.

GARCIA, S. *A Systematic Requirements Engineering Approach for Decision Support Systems*. Barcelona, Spain, 2014. Citado na página 31.

IBM RATIONAL DOORS. *Get It Right The First Time*. [S.l.], 2010. Citado 2 vezes nas páginas 31 e 32.

JR, P. L. R. L. *Requisitos de Métodos de Rastreabilidade entre os Requisitos e o Código de Software*. [S.l.], 2007. Citado na página 31.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. [S.l.], 2003. Citado 3 vezes nas páginas 17, 18 e 23.

LEFFINGWELL, D. *Agile Software Requirements*. [S.l.], 2010. Citado 8 vezes nas páginas 5, 18, 21, 22, 23, 24, 29 e 32.

MC. *Institucional - O ministério*. [S.l.], 2009. Disponível em: <http://www.mc.gov.br/institucional/>. Citado na página 15.

SCALED AGILE INC. *Release Management*. [S.l.], 2015. Disponível em: <http://www.scaledagileframework.com/release-management/>. Citado na página 24.

Apêndices

APÊNDICE A – Primeiro Apêndice

Texto do primeiro apêndice.

APÊNDICE B – Segundo Apêndice

Texto do segundo apêndice.

Anexos

ANEXO A – Primeiro Anexo

Texto do primeiro anexo.

ANEXO B – Segundo Anexo

Texto do segundo anexo.