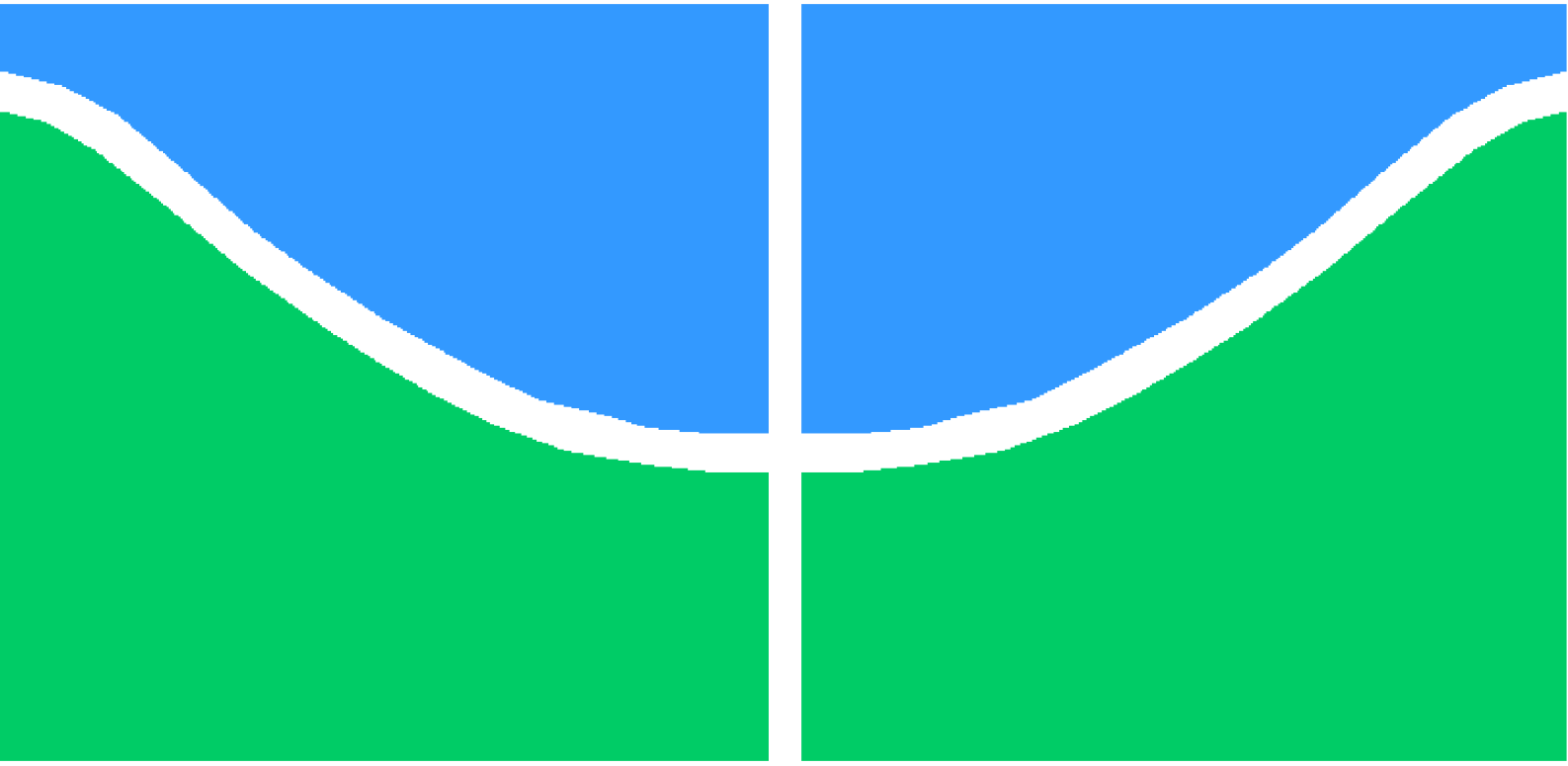


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Requisitos de Software - 201308

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
CMMI	Capability Maturity Model - Integration
MPS.BR	Melhoria do Processo de Software Brasileiro

Sumário

1 Introdução

A Engenharia de Requisitos é uma das etapas existentes no contexto de Engenharia de Software. É tida como um dos processos essencias e que mais afetam o desenvolvimento de um sistema (HOFMANN H.F. e LEHNER, 2001), e é o ponto no qual o domínio do problema deve ser entendido já que é nela que ocorre a aquisição, refinamento e verificação das necessidades do cliente para um sistema de software (IEEE, 1984).

De acordo com esses fatos, esse trabalho tem por objetivo a elaboração de uma solução de um problema no contexto do Ministério das Comunicações, utilizando o processo de Engenharia de Requisitos como base. Além disso, em conjunto, será utilizado uma abordagem e modelos de maturidade.

1.1 Visão Geral do Relatório

O relatório será composto por capítulos, onde cada capítulo descreve e explica assuntos marcantes. Os capítulos são:

1.1.1 Introdução

Introdução do relatório, descreve brevemente como iremos abordar certos assuntos, qual o objetivo do trabalho e alguns conceitos.

1.1.2 Contexto

Descreve de maneira geral o contexto da Empresa e do problema a ser solucionado.

1.1.3 Escolha da Abordagem

Descreve um pouco sobre as duas abordagens (adaptativa e tradicional), justifica e faz uma escolha de abordagem.

1.1.4 Modelos de Maturidade

Fala um pouco sobre a definição de Modelos de Maturidade, e como eles foram utilizados no contexto do projeto.

1.1.5 Engenharia de Requisitos

Mostra todo o processo de Engenharia de Requisitos a ser utilizado no projeto (papéis, atividades, entre outras coisas).

1.1.6 Planejamento do projeto

Diz de forma geral como será dado o planejamento nos diferentes níveis do projeto e mostra o andamento do cronograma

1.1.7 Técnicas de Elicitação de Requisitos

Fala brevemente sobre diferentes técnicas de requisitos, quais vão e não vão ser utilizadas e o motivo desta escolha.

1.1.8 Tópicos de Gerenciamento de Requisitos

Fala como será dado a gerência de requisitos no contexto do trabalho, quais atributos serão utilizados para os requisitos, e como será a rastreabilidade.

1.1.9 Ferramentas de Gerência de Requisitos

Mostra um *overview* sobre três ferramentas, a escolha de uma delas, e a justificativa desta escolha.

1.1.10 Considerações Finais

Considerações finais sobre o trabalho em si, e as conclusões obtidas.

1.1.11 Referências

Traz a fonte de algumas informações utilizadas como base para o 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 processo 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, 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 de Requisitos de Software, responsável por elaborar uma solução para o problema da 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, 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 *Scrum Master* 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 Modelos de Maturidade

Modelos de Maturidade, nos dias de hoje, ajudam organizações a melhorarem a maneira com que elas fazem negócio. No contexto de *Engenharia de Software*, estes modelos ajudam organizações a desenvolver e manter a qualidade dos seus produtos e serviços, trazendo um conjunto de boas práticas a serem seguidas para se atingir determinado nível de maturidade (SEI, 2010).

No contexto deste projeto, os modelos de maturidade (em especial o CMMI e o MPS.BR) foram utilizados como forma de apoio durante no projeto. Entre essas formas, a principal foi na definição de tarefas que seriam utilizadas no processo de Engenharia de Requisitos. Esperava-se que pelo menos as áreas de processo relacionadas a área de requisitos (*REQM - Requirements Management, e RD - Requirements Development*) tivessem todas as suas práticas presentes no projeto, direta ou indiretamente, então, foi feita essa relação entre o modelo de maturidade, a abordagem escolhida (adaptativa - SAFe) e o projeto em si.

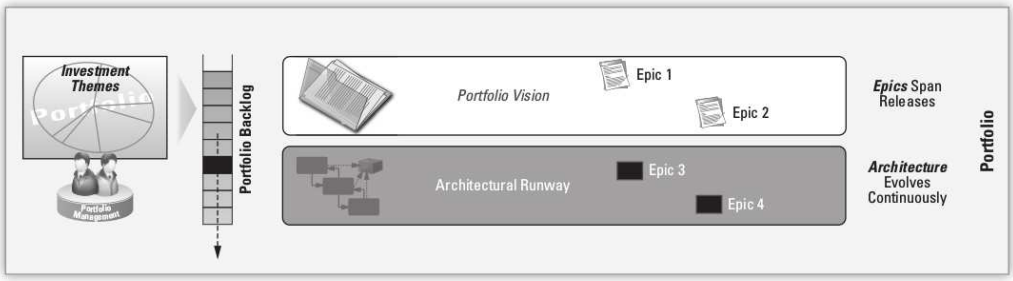
5 Engenharia de Requisitos

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.

5.1 Scaled Agile Framework

5.1.1 Nível de Portfólio

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

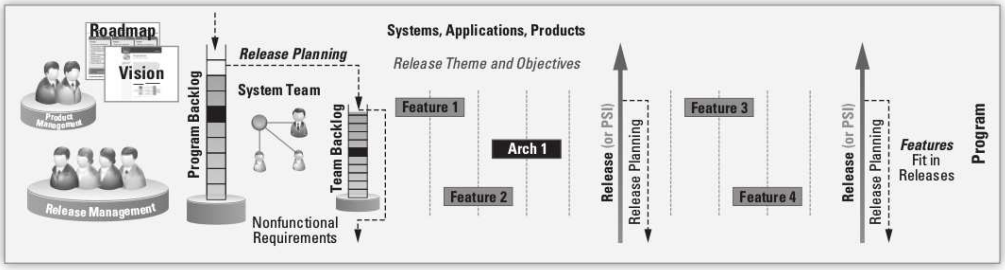


Responsável pelos artefatos *epics* (épicos) e *investiment themes* (temas de investimento), este nível de projeto traz os requisitos com maior nível de abstração dentre os três níveis. Contém um tipo de time (*portfolio management team*), costuma ter seu próprio backlog (*portfolio backlog*), e apresenta 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 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, e também traria a necessidade de gerenciar mais

um outro *backlog*. Logo preferiu uni-lo ao *backlog* do Nível de Programa, deixando um papel cuidando de ambos.

5.1.2 Nível de Programa

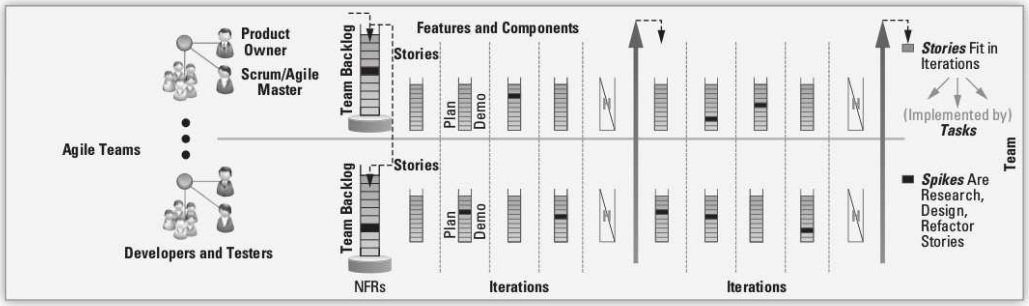
Figura 2: Nível de Programa no SAFe - (LEFFINGWELL, 2010, p. 39)



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.

5.1.3 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* (PO), 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). As estórias serão então validadas pelos POs (alunos de MPR).

5.2 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.

5.2.1 Papéis no Nível de Portfólio

5.2.1.1 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 por elaborar e apresentar épicos, e será composto somente de alunos de Requisitos.

5.2.2 Papéis no Nível de Programa

5.2.2.1 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) não será utilizado e algumas de suas tarefas serão atribuidas a outros papéis, pois uma de suas principais atribuições é a de gerenciar os diversos PO's, quando existem. Como no contexto do grupo haverá não haverão vários POs, as tarefas do PM serão atribuidas ao PO, e este papel deixará de existir.

5.2.2.2 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., 2015a). 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 deverá ser *release manager*.

5.2.2.3 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., 2015a). No contexto do projeto, diferente da bibliografia, eles não serão responsáveis pelo desenvolvimento e implantação do *software*, e será praticado pelo dono do contexto (um aluno de MPR).

5.2.2.4 Release Train Engineer

O RTE é quase que o facilitador do Nível de Programa. Normalmente tem *background* na área de gerência de programa, projeto ou desenvolvimento (SCALED AGILE INC., 2015b). No contexto do grupo 1, o RTE funcionará como *"O Scrum Master do Nível de Programa"*, além de ajudar a escrever as *features* junto com o PO, e gerenciar o Nível de Programa (cuidará do *backlog* de programa), também será o papel que documentará as *features* no Documento de Visão e cuidará do *roadmap*. Este papel será composto de todos os alunos de Requisitos.

5.2.3 Papéis no Nível de Time

5.2.3.1 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* irá ajudar o time de desenvolvimento a escrever as estórias. O papel de PO será delegado a integrantes de MPR, que deverão validar e analisar as estórias implementadas pelos integrantes do grupo de Requisitos.

5.2.3.2 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*.

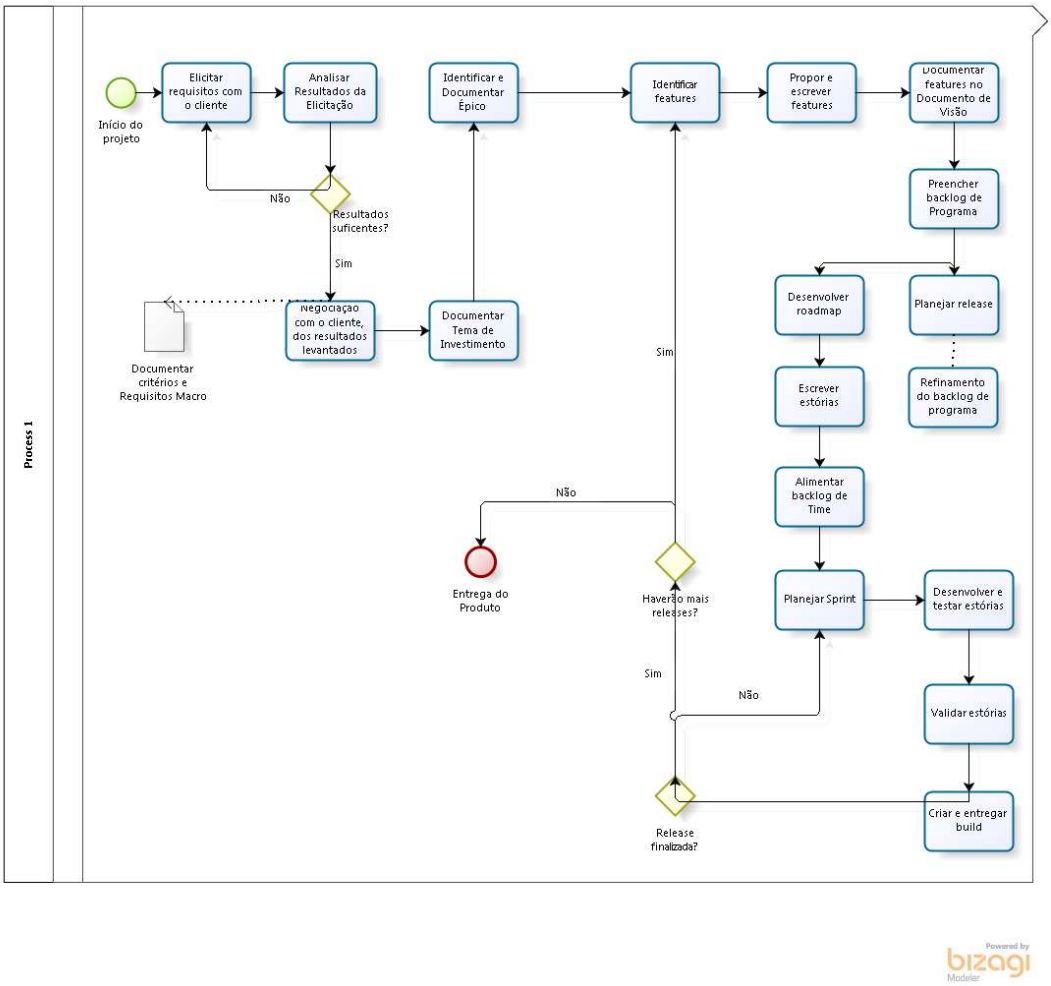
5.2.3.3 Desenvolvedores e Testadores

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

5.3 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:

5.3.1 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.

5.3.2 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.

5.3.3 Documentar Tema de Investimento

Atividade feita pelos *epic owners*, consiste em se documentar o tema de investimento que guiará a instituição. O resultado será armazenado no *program backlog*.

5.3.4 Gerenciar Tema de Investimento

Atividade feita pelo *epic owner*, consiste em se encontrar inconsistências nos temas de investimento documentados no *program backlog*, garantir a rastreabilidade, e garantir acesso ao *log* de todas as mudanças ocorridas em um determinado tema de investimento. Essa atividade é de suporte, e irá acontecer quase que sempre que um determinado tema de investimento sofrer mudança.

5.3.5 Identificar épico

Atividade feita pelos *epic owners* e *business owner*, consiste em se identificar épicos nas necessidades e descrições dadas do problema até o momento.

5.3.6 Preparar épico

Atividade feita pelos *epic owners*, consiste em preparar formalmente um determinado épico identificado.

5.3.7 Apresentar épico

Atividade feita pelos *epic owners*, consistido em apresentar o épico para o *business owner*.

5.3.8 Decisão a respeito do épico

Atividade feita pelo *business owner*, consiste em dar uma decisão final sobre um épico apresentado. A decisão pode ser de aprovação ou reprovação de um determinado épico.

5.3.9 Documentar épico

Atividade feita pelos *epic owners*, consiste em se documentar determinado épico.

5.3.10 Gerenciar épico

Atividade feita pelo *epic owner*, consiste em se encontrar inconsistências nos épicos documentados no *program backlog*, garantir a rastreabilidade, e garantir acesso ao *log* de

todas as mudanças ocorridas em um determinado épico. Essa atividade é de suporte, e irá acontecer quase que sempre que um determinado épico sofrer mudança.

5.3.11 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.

5.3.12 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.

5.3.13 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).

5.3.14 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.

5.3.15 Documentar *features* no Documento de Visão

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

5.3.16 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.

5.3.17 Gerenciar *features*

Atividade feita pelo RTE, consiste em se encontrar inconsistências nas *features* documentadas no Documento de Visão e *program backlog*, garantir a rastreabilidade destas

features, e garantir acesso ao *log* de todas as mudanças ocorridas em uma determinada *feature*. Essa atividade é de suporte, e irá acontecer quase que sempre que uma determinada *feature* sofrer mudança.

5.3.18 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.

5.3.19 Documentar e gerenciar *roadmap*

Feita pelo RTE, consiste em documentar, gerenciar e criar (se necessário) o *roadmap* da *release*.

5.3.20 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.

5.3.21 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.

5.3.22 Alimentar *backlog* de time

Consiste em documentar e armazenar o conjunto de estórias no backlog, garantindo também sua rastreabilidade. Feito pelo Scrum Master.

5.3.23 Gerenciar *backlog* de time

Atividade feita pelo Scrum Master, consiste em se encontrar inconsistências nas estórias de usuário, *spikes* e estórias técnicas documentadas *backlog* de time, garantir a rastreabilidade destas atividades, e garantir acesso ao *log* de todas as mudanças ocorridas. Essa atividade é de suporte, e irá acontecer quase sempre que determinada mudança ocorrer no *backlog* de time.

5.3.24 Documentar e identificar tarefas (*tasks*)

Atividade feita pelo time de desenvolvimento, consiste em identificar pequenas tarefas dentro de uma estória de usuário, estória técnica ou *spike*, e documentar essas tarefas no *backlog* de time.

5.3.25 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.

5.3.26 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.

5.3.27 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.

5.3.28 Validar estória

Feita pelo PO, consiste em dar um veredito sobre determinada estória implementada e testada. Será feita com base nos critérios de aceitação.

6 Planejamento do Projeto

Neste capítulo será explicado o planejamento utilizado pelo Grupo 1, e justificativas sobre certas decisões.

6.1 SAFe

O SAFe traz tipos de planejamento diferentes para cada nível de projeto. Há o planejamento do épico, o planejamento da *release*, o planejamento da *sprint*, entre outros.

6.1.1 Planejamento no Nível de Portfólio

O planejamento neste nível será baseado no planejamento de um determinado *épico*, organizo pelos *Epic Owners*. Além disso, o gerenciamento neste nível de projeto também será feito pelo *epic owner*.

6.1.2 Planejamento no Nível de Programa

O planejamento no Nível de Programa será baseado no planejamento da *release*, organizada pelo RTE no contexto do grupo 1. Além disso, para ajudar no planejamento, haverá o *Roadmap*, que será atualizado após o *release planning*.

6.1.3 Planejamento no Nível de Time

O planejamento no Nível de Time será baseado no planejamento da *sprint*, organizada pelo Scrum Master, influenciada pelos Desenvolvedores e Product Owners no contexto do grupo 1. Haverá um *kanban* deste nível mostrando entregas a serem feitas, o que está sendo feito, etc. Após as *sprints* ocorrerá a retrospectiva, e seus resultados irão influenciar no planejamento da próxima *sprint*.

6.2 Cronograma

O cronograma ajudará as equipes fornecendo uma visão sobre as tarefas a serem desenvolvidas, o pré-requisito dessas tarefas, e o prazo disponível. Os cronogramas de MPR e Requisitos embora não estejam atrelados entre si, estão conectados de diversas maneiras. Exemplo: certas atividades de Requisitos tem como pré-requisitos atividades de MPR, e este *link* é demonstrado. Certas atividade de Requisitos necessitam ser completadas também por membros de MPR, o que também é demonstrado (designação de tarefa).

O cronograma foi feito no *software* Gantter, que se adequou muito bem ao time por ser facilmente gerenciado, ter um bom *track* de ações, ser em nuvem, seguro, entre outras qualidades.

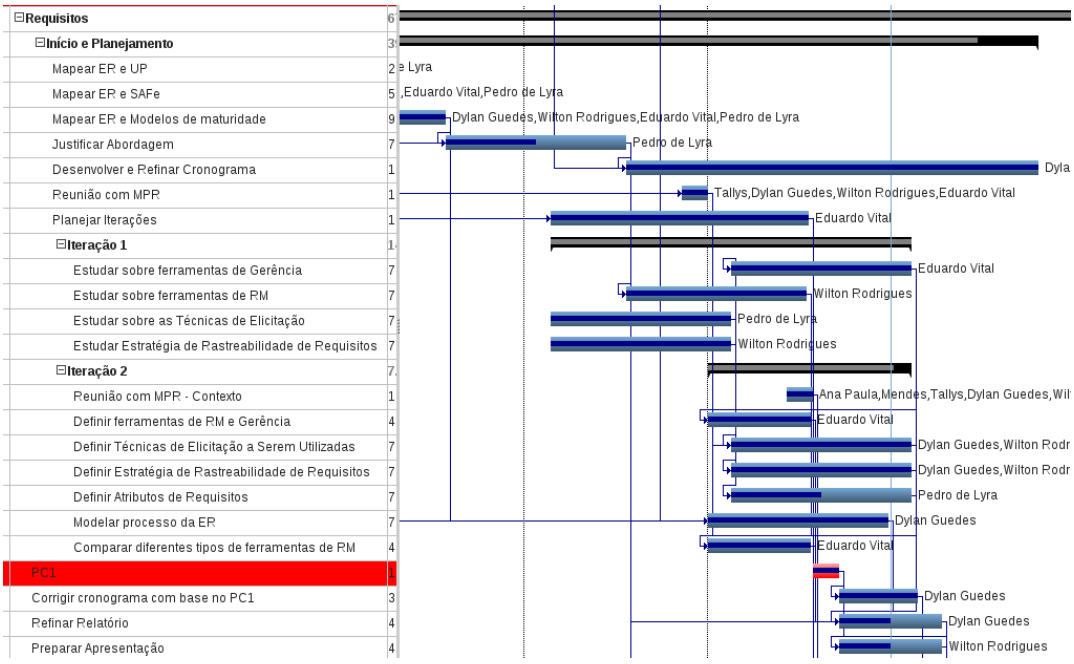
Embora no cronograma pudesse ser especulado possíveis datas para tarefas futuras (exemplo: futuros planejamentos de *release*), tais especulações ainda não são mostradas, pois julga-se como muito cedo. Além disso, são marcados no cronograma marcos importantes do projeto, como por exemplo, o PC, data de entrega do trabalho, entre outros.

Cronograma:

Figura 5: Cronograma da equipe de Requisitos, feito no *Gantter*

		Name	Duration	Start	Finish	Predecessors	Resources
32		**Início e Planejamento**	39.28d?	**31/03/2015**	**08/05/2015**		
33		Mapear ER e UP	2.04d	31/03/2015	02/04/2015		Dylan Guedes,Wilton Rodrigues,Eduardo Vital,Pedro de Lyra
34		Mapear ER e SAFe	5d	02/04/2015	07/04/2015		Dylan Guedes,Wilton Rodrigues,Eduardo Vital,Pedro de Lyra
35		Mapear ER e Modelos de maturidade	9d	07/04/2015	16/04/2015		Dylan Guedes,Wilton Rodrigues,Eduardo Vital,Pedro de Lyra
36		Justificar Abordagem	7d?	16/04/2015	22/04/2015	33,34,35	Pedro de Lyra
37		Desenvolver e Refinar Cronograma	16d	22/04/2015	08/05/2015	5,36	Dylan Guedes
38		Reunião com MPR	1d?	25/04/2015	25/04/2015	2,3	Tallys,Dylan Guedes,Wilton Rodrigues,Eduardo Vital
39		Planejar Iterações	10d?	20/04/2015	29/04/2015	3	Eduardo Vital
40		**Iteração 1**	14d?	**20/04/2015**	**03/05/2015**		
41		Estudar sobre ferramentas de Gerência	7d?	26/04/2015	03/05/2015	44	Eduardo Vital
42		Estudar sobre ferramentas de RM	7d?	22/04/2015	29/04/2015	36	Wilton Rodrigues
43		Estudar sobre as Técnicas de Elicitação	7d?	20/04/2015	26/04/2015		Pedro de Lyra
44		Estudar Estratégia de Rastreabilidade de Requisitos	7d?	20/04/2015	26/04/2015		Wilton Rodrigues
45		**Iteração 2**	7.89d?	**26/04/2015**	**03/05/2015**		
46		Reunião com MPR - Contexto	1d?	29/04/2015	29/04/2015		Ana Paula,Mendes,Tallys,Dylan Guedes,Wilton Rodrigues,Ed
47		Definir ferramentas de RM e Gerência	4d?	26/04/2015	29/04/2015	38,41,42,49,50	Eduardo Vital
48		Definir Técnicas de Elicitação a Serem Utilizadas	7d?	26/04/2015	03/05/2015	38,43	Dylan Guedes,Wilton Rodrigues,Eduardo Vital,Pedro de Lyra
49		Definir Estratégia de Rastreabilidade de Requisitos	7d?	26/04/2015	03/05/2015	44	Dylan Guedes,Wilton Rodrigues,Eduardo Vital,Pedro de Lyra
50		Definir Atributos de Requisitos	7d?	26/04/2015	03/05/2015	44	Pedro de Lyra
51		Modelar processo da ER	7d?	26/04/2015	02/05/2015	10,11,34,35,36	Dylan Guedes
52		Comparar diferentes tipos de ferramentas de RM	4d?	26/04/2015	29/04/2015	38,41,42	Eduardo Vital
53		PC1	1d?	30/04/2015	30/04/2015		
54		Corrigir cronograma com base no PC1	3.05d?	01/05/2015	04/05/2015	53	Dylan Guedes
55		Refinar Relatório	4d?	01/05/2015	04/05/2015	36,46,47,48,49,5	Dylan Guedes
56		Preparar Apresentação	4d?	01/05/2015	04/05/2015	53,55	Wilton Rodrigues
57		Entrega Relatorio	0d?	04/05/2015	04/05/2015	46,54,55	Dylan Guedes
58		Apresentação	2.04d?	11/05/2015	14/05/2015	55,56	Dylan Guedes,Wilton Rodrigues,Eduardo Vital,Pedro de Lyra

Figura 6: Status das atividades de Requisitos, feito no *Gantter*



7 Técnicas de Elicitação de Requisitos

Em determinados pontos do ciclo de vida de um projeto, a organização precisará obter uma certa compreensão das condições que o sistema deverá satisfazer, questionamentos como "Quais problemas o sistema pretende solucionar?", "Qual público alvo ele pretende servir?", "Quais suas restrições?"irão emergir e serão respondidos pela definição dos requisitos realizada pela equipe.

Este processo de buscar respostas para as questões levantadas é conhecido como descoberta de requisitos. A equipe procura formas de identificar e descrever as características e capacidades do *software* que se propõe a solucionar um problema e para isso é necessário que o time foque em manter a concordância, compreensão e entendimento entre todos os membros para que todos compartilhem uma visão do sistema que será desenvolvido.

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).

Considerando a importância da descoberta de requisitos, várias técnicas são propostas para auxiliar e otimizar o processo de elicitação. Cada uma delas possui suas vantagens e desvantagens e uma combinação delas pode ser necessária para atender as circunstâncias de um determinado projeto. Neste documento, serão apresentadas sete técnicas, sendo três destas escolhidas pela equipe para serem utilizadas no projeto:

7.1 Workshop de Requisitos

O principal objetivo desta técnica é estabelecer um consenso entre os requisitos do sistema em um curto espaço de tempo. Para isso, ele indica que os principais *stakeholders* do projeto devem ser reunidos com os integrantes da equipe que participarão do *workshop* (Product Owner, Product Manager, um membro do time, etc...). O *workshop* procura manter uma concordância à respeito dos requisitos entre os envolvidos no projeto através da exposição de ideias de todos os participantes, ele geralmente dura um dia e envolve uma introdução para informar aos participantes como o *workshop* funcionará, uma contextualização para descrever o estado atual do projeto, um *brainstorming* para que todos os participantes possam expor suas ideias e discutir à respeito do projeto, um almoço para servir de intervalo (apesar de que durante o almoço é recomendado que a discussão mantenha de forma saudável para que o clima do workshop não se perca), outro *brains-*

torming após o almoço, definição de *features* para identificar requisitos de alto nível do sistema, priorização de algumas das *features* propostas e para encerrar, um resumo sobre tudo que ocorreu durante o dia.

Esta técnica não será utilizada pois o esforço e tempo exigido é muito grande e não seria possível, e, além disso, é uma técnica relativamente massante.

7.2 Brainstorming

Esta técnica procura oferecer uma oportunidade para que todos os participantes exponham suas ideias em relação ao que imaginam ser uma solução potencial para o projeto. O Brainstorming é dividido em duas fases: geração de ideias e redução de ideias. Durante a geração de ideias, os participantes identificam a maior quantidade de ideias possíveis, sendo que estas propõe possíveis soluções, capacidades, restrições e características do sistema. Durante a redução de ideias, os participantes realizam a análise das ideias propostas, isso inclui um refinamento, uma priorização, uma organização, uma triagem, etc. A principal proposta do Brainstorming é criar um ambiente que seja capaz de receber discussões saudáveis, ele incentiva que os participantes se abram e falem o que pensam e não permite críticas destrutivas já que isso desmotivaria os participantes e os colocariam em uma posição desconfortável. Esta técnica valoriza a criatividade e aposta na combinação das ideias geradas pelos participantes.

Essa técnica será utilizada pois é efetiva e flexível, que é o perfil de técnica que melhor se adequa ao time.

7.3 Entrevistas e Questionários

Esta técnica consiste na realização de diversos questionários durante uma entrevista com um *stakeholder*. É sugerido que o entrevistador responsável por interagir com o cliente priorize perguntas livres de contexto primeiramente, para evitar a realização perguntas tendenciosas que possam influenciar a resposta do cliente. Após a realização das perguntas livres de contexto, é recomendado que se faça perguntas que explorem possíveis soluções, para isso, é interessante que o entrevistador compreenda o contexto do projeto, o que pode exigir um estudo. Após entrevistar os principais *stakeholders* do projeto, as respostas serão analisadas e irão produzir uma certa convergência à respeito do entendimento do problema.

7.4 Mock-Ups

A proposta desta técnica é construir um protótipo da interface de usuário referente à um requisito, para que o cliente colabore com a extração das funcionalidades contidas naquela interface. Isso permite que o cliente demonstre as possíveis interações imaginadas com o sistema antes que qualquer código seja implementado. É possível desenhar no papel a interface imaginada ou utilizar ferramentas que automatizem o processo e construam *wireframes*.

7.5 Análise Competitiva

A análise competitiva é uma técnica que procura identificar possíveis concorrentes e estudar o que eles oferecem em seus produtos. À partir deste estudo, a equipe realiza uma avaliação para determinar funcionalidades potenciais do concorrente que podem ser incluídas no projeto por refletirem o interesse dos clientes.

7.6 Sistema de Solicitação de Mudanças do Cliente

Esta técnica consiste na implementação de um sistema que aceite sugestões dos usuários à respeito de possíveis evoluções e correções no *software*. É basicamente um sistema de *feedbacks* da experiência obtida por um usuário ao utilizar o *software*.

7.7 Modelagem caso-de-uso

Apesar das equipes ágeis utilizarem primariamente estórias de usuário para descrever os requisitos de um sistema, em muitas vezes, esta abordagem se torna inadequada devido à complexidade do *software*. Em situações em que o *software* é composto por vários sistemas e componentes, o relacionamento entre eles pode se tornar difícil de descrever em estórias de usuário e os casos de uso podem ser mais adequados para especificar a interação entre o *software* e os seus componentes (sub-sistemas, dispositivos de hardware, entre outros).

7.8 Escolha das Tecnicas

Assim sendo, a equipe optou pelas seguintes técnicas de elicitação:

- Mock-ups: Imaginamos que a construção de protótipos que representem a estrutura das interfaces de usuário da nossa aplicação nos ajude a extrair, com a colaboração

do cliente, as funcionalidades presentes naquele ponto do *software*, já que o cliente indicará as possíveis interações que pretende ter com aquela parte do sistema.

- Brainstorming: Avaliamos ser de fundamental importância a troca de ideias entre os envolvidos no projeto, para que toda a informação resultante do que foi imaginado por cada um convirja para um conjunto de características e capacidades do sistema que reflitam as necessidades do cliente.

- Análise Competitiva: Consideramos a análise de soluções oferecidas por concorrentes de sucesso uma excelente abordagem para avaliar funcionalidades que possam ser incluídas ao nosso projeto. Soluções de sucesso que resolvem problemas parecidos com o nosso e oferecem um produto relativamente próximo à solução imaginada pode ser uma fonte de ideias para extrair as capacidades do sistema a ser desenvolvido.

8 Tópicos de Gerenciamento de Requisitos

8.1 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 *Requirements Management* 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.

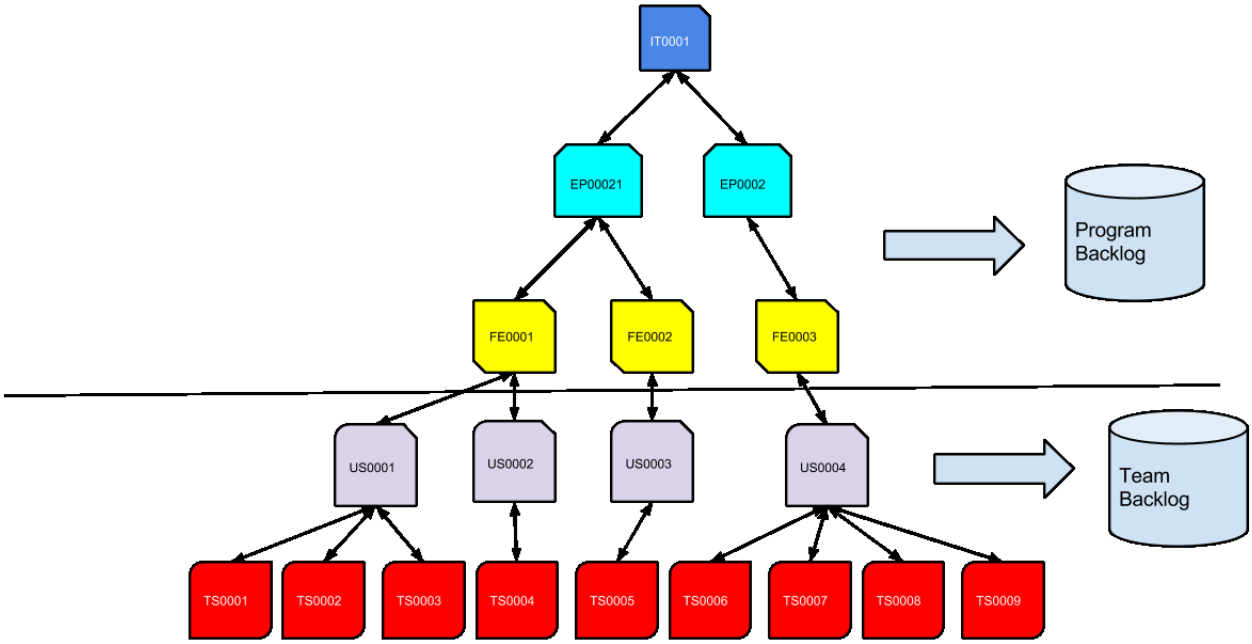
A nomenclatura utilizada será uma concatenação entre prefixo (mostrados abaixos) e prefixos (número do índice do item). Lista de prefixos utilizados:

Tabela 1: Esquema de IDs dos Requisitos

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

Esquema da rastreabilidade:

Figura 7: Esquema de rastreabilidade de requisitos que será utilizado



8.2 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 *WSJF - 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 *WSJF* 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).

Ou seja, 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 (WSJF), será assumido valor 3 para alto, 2 para médio e 1 para baixo para os atributos correspodentes. Exemplo: Um requisito com custo do atraso alto e esforço médio:

$$Peso = C/E = 3/2 = 1.5 \tag{8.1}$$

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

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

$$Peso = 1/3 = 0.3333 \tag{8.2}$$

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

9 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 observado 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 com justificativa.

9.1 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: features, releases, sprints, épicos, estórias de usuário, pontos da estória, cenários de uso, e casos de teste

- Rastreabilidade dos requisitos
- Implementação nativa de alguns atributos de 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 apresenta uma excelente capacidade de rastreabilidade relacionando os épicos com as estórias de usuários e com os casos de testes, funcionando em ambos os sentidos (para frente e para trás), e ainda possibilita a criação hierárquica dos requisitos, de forma extremamente eficiente.

São implementados os seguintes atributos de requisitos de forma nativa:

- Fonte
- Prioridade
- Comentários
- Risco
- Prazo

Os demais atributos podem ser implementados por marcadores ou dentro da descrição do requisito, que permite a inserção de texto formatado.

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.

Uma interessante característica do Jama é permitir a criação de requisitos personalizados, o que possibilita a criação de temas de investimentos dentro da ferramenta.

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.

9.2 Ferramenta 2 - Rally

- Suporte à múltiplos projetos
- Suporte à múltiplas equipes e papéis
- Interface prática, amigável e organizada
- Rastreabilidade dos requisitos
- Implementação nativa de alguns atributos de 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.

São implementados os seguintes atributos de requisitos de forma nativa:

- Comentários
- Status
- Prazo
- Nível de teste
- Esforço

Os demais atributos podem ser implementados por marcadores, mas a visualização desses marcadores não é tão prática, também podem ser implementados dentro da descrição do requisito, que permite a inserção de texto formatado.

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.

9.3 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.

São implementados os seguintes atributos de requisitos de forma nativa:

- Fonte
- Prazo

- Prioridade
- Comentários

Os demais atributos podem ser implementados dentro do corpo do requisito, que é um documento, onde praticamente tudo é passível de personalização. No entanto isso se mostra um processo complicado.

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, possuindo uma personalização incrível, embora igualmente trabalhosa.

9.4 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. E embora a ferrramenta apresente muitos atributos de requisitos implementados por *default*, os demais atríbutos não são tão úteis quando implementados via marcadores, sendo difíceis de gerenciar.

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. Seu alto poder de personalização é algo notável, mas essa customização é custosa e demasiadamente trabalhosa, tendo seus benefícios anulados. Ainda pesam contra a ferramenta: o fato de não ter ser nativamente 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. Por esses motivos também optou-se por não utilizá-la.

Sendo assim, optou-se pela ferramenta Jama. Poderia-se dizer que foi a alternativa restante, 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, com criação hierárquica de requisitos, e efetiva implementação não-nativa de atríbutos de requisitos.

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.

10 Considerações Finais

Ao fim da elaboração desse primeiro relatório, é possível constatar como o processo de Engenharia de Requisitos é essencial para a produção eficaz de *software*. Processo esse tão importante que requisitos deficientes são a maior causa de falhas nos projetos de *software* (HOFMANN H.F. e LEHNER, 2001). Dessa forma concluimos que a Engenharia de Requisitos deve ser uma etapa do processo de desenvolvimento de *software* que deve ser feita com um alto nível de qualidade, pois o custo de para se corrigir um erro aumenta significativamente ao longo do tempo (BOEHM, 1981). Ou seja, quanto maior a qualidade e empenho da equipe nesta etapa do processo, menores serão os custos desnecessários, e maiores serão as chances de sucesso.

Referências

BEATTY, J. *Winning the Hidden Battle: Requirements Tool Selection and Adoption*. Austin, TX USA, 2013. Citado na página 47.

BOEHM, B. W. *Software Engineering Economics*. [S.l.], 1981. Citado na página 53.

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 19.

GARCIA, S. *A Systematic Requirements Engineering Approach for Decision Support Systems*. Barcelona, Spain, 2014. Citado na página 43.

HOFMANN H.F. e LEHNER, F. *Requirements Engineering as a Sucess Factor in Software Projects*. [S.l.], 2001. v. 18, 58-66 p. Citado 2 vezes nas páginas 15 e 53.

IBM RATIONAL DOORS. *Get It Right The First Time*. [S.l.], 2010. Citado na página 44.

JR, P. L. R. L. *Requisitos de Métodos de Rastreabilidade entre os Requisitos e o Código de Software*. [S.l.], 2007. Citado na página 43.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. [S.l.], 2003. Citado 3 vezes nas páginas 19, 20 e 27.

LEFFINGWELL, D. *Agile Software Requirements*. [S.l.], 2010. Citado 9 vezes nas páginas 5, 20, 25, 26, 27, 28, 29, 39 e 44.

MINISTéRIO DAS COMUNICAçõES. *Institucional - O ministério*. [S.l.], 2009. Disponível em: <http://www.mc.gov.br/institucional/>. Citado na página 17.

SCALED AGILE INC. *Release Management*. [S.l.], 2015. Disponível em: <http://www.scaledagileframework.com/release-management/>. Citado na página 28.

SCALED AGILE INC. *RTE*. [S.l.], 2015. Disponível em: <http://www.scaledagileframework.com/rte/>. Citado na página 28.

SOFTWARE ENGINEERING INSTITUTE. *CMMI for Development, 1.3*. [S.l.], 2010. Disponível em: <http://www.sei.cmu.edu/reports/10tr033.pdf>. Citado na página 23.

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. *IEEE Guide to Software Requirements Specification*. [S.l.], 1984. Citado na página 15.