Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma abordagem usando features BDD e Modelo de Objetivos para o desenvolvimento ágil de software.

Fábio Barros Leal

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Genaina Nunes Rodrigues

Brasília
2019

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

Barros Leal, Fábio
Ba Uma abordagem usando features BDD e Modelo de Objetivos para o desenvolvimento ágil de software / Fábio Barros Leal; orientador Genaina Nunes Rodrigues. -- Brasília, 2019.
84 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2019.

1. Engenharia de Requisitos. 2. Desenvolvimento Guiado por Comportamento. 3. Engenharia de Requisitos Orientada por Objetivos. 4. Projetos Ágeis. 5. Gestão de Projetos Ágeis. I. Nunes Rodrigues, Genaina, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma abordagem usando features BDD e Modelo de Objetivos para o desenvolvimento ágil de software.

Fábio Barros Leal

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Genaina Nunes Rodrigues (Orientador)
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
CIC/UnB

Prof. Dr. João Henrique Correia Pimentel
UFPE

Prof. Dr. Aletéia Patrícia Favacho de Araújo
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 09 de Agosto de 2019

Dedicatória

À minha esposa, que esteve ao meu lado na consecução desse trabalho, me dando forças e fôlego para não esmorecer. Seu auxílio foi contumaz importante, motivando e despertando a vontade de vencer. Sua dedicação e atenção nos momentos mais críticos me inspirou os pensamentos de uma forma bem especial, levando-me a novos horizontes no despertar do caminho do saber.

Agradecimentos

Em primeiro lugar a Deus, que é o princípio de tudo e causa primária de todas as coisas. À minha esposa, que suportou a minha ausência, fruto de horas e meses na elaboração desta obra.

Aos meus professores de mestrado que me fizeram uma nova pessoa, em especial a minha orientadora, com a sua paciência, sendo educadora, compreensiva, e às vezes rígida nos momentos em que houve a necessidade, me conduzindo ao caminho certo da aprendizagem.

Agradeço também a todos os meus amigos que me deram uma força para que eu pudesse levar a cabo este projeto, pelos incentivos e pela credibilidade.

Resumo

O *Behavior-Driven Development* (BDD) centra-se na colaboração e descoberta do comportamento do sistema incorporando a definição de funcionalidades por meio de *features* baseadas em especificação por exemplos, o denominado *Specification by Example*. Por outro lado, o *Goal-Oriented Requirements Engineering* (GORE) utiliza-se de objetivos para capturar as intencionalidades do sistema em diferentes níveis de abstração, a fim de especificar, estruturar, analisar, negociar, documentar e modificar requisitos para serem efetivamente alcançados. Portanto, o GORE tem como meta utilizar o conceito de objetivo para dar suporte às fases iniciais da engenharia de requisitos.

Deste modo, aproveitando o benefício das duas abordagens, o trabalho proposto tem como objetivo propor um método para apoiar a gestão de projetos ágeis de software a partir das técnicas BDD e GORE. Tal método propõe a combinação das técnica BDD e o uso dos modelos GORE para contribuir no gerenciamento dos requisitos e como estes realizam os seus respectivos objetivos de negócio. Adicionalmente, propomos um algoritmo para analisar qualitativa e quantitativamente a exequibilidade dos objetivos do modelo de objetivos na presença *tasks* seguindo o conceito de *living documentation* da abordagem BDD, e permitindo a rastreabilidade entre requisitos e objetivos de negócio e suas realizações de forma dinâmica. Desta forma, a nossa abordagem de modelagem baseada na associação das *features* BDD a *tasks* pode abrir caminho para contribuir com a gestão de projetos ágeis.

O método proposto foi implementado na ferramenta piStar e analisado no Sistema de Boletins e Alterações (SISBOL) de forma a avaliar a viabilidade da proposta no apoio a gestão de projetos de software ágeis. Nesse contexto, foi utilizado um método para coleta dos dados das especificações de requisitos do projeto PROMISE-EB. Por meio da coleta de dados das 16 *sprints* do projeto foi possível monitorar percentualmente o grau de aceitação das *features*, bem como a respectiva realização dos objetivos de negócio.

Palavras-chave: Requisitos, Desenvolvimento Guiado por Comportamento, Engenharia de Requisitos Orientada por Objetivos, Projetos Ágeis, Gestão de Projetos Ágeis

Abstract

Behavior-Driven Development (BDD) focuses on the collaboration and discovery of system behavior incorporating the definition of functionalities through features based on Specification by Example. On the other hand, the Goal-Oriented Requirements Engineering (GORE) uses goals to capture the intentionalities of the system in different levels of abstraction, in order to specify, structure, analyze, negotiate, document and modify requirements to be effectively achieved. Therefore, GORE aims to use the concept of objective to support the initial phases of requirements engineering.

Thus, taking advantage of the benefit of both approaches, the proposed work aims to propose a method to support the management of agile projects based on the BDD and GORE techniques. This method proposes the combination of BDD techniques and the use of GORE models to contribute to the management of requirements and how they achieve their respective business objectives. Additionally, we propose an algorithm to analyze qualitatively and quantitatively the feasibility of the objectives of the objectives model in the presence of tasks following the concept of living documentation of the BDD approach, and allowing traceability between requirements and business objectives and their achievement in a dynamic way. In this way, our modeling approach based on the association of BDD features with tasks can open the way to contribute to agile project management.

The proposed method was implemented in the piStar tool and analyzed in the System of Bulletins and Changes (SISBOL) in order to assess the feasibility of the proposal in supporting the management of agile software projects. In this context, a method was used to collect data from the requirements specifications of the PROMISE-EB project. Through the data collection of the 16 sprints of the project it was possible to monitor percentually the degree of acceptance of the features, as well as the respective achievement of the business objectives.

Keywords: Requirements, Behavior-Driven Development (BDD), Goal-Oriented Requirements Engineering (GORE), Agile Project, Agile Project Management

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

O processo de desenvolvimento de software seguindo metodologias ágeis pode envolver algumas peculiaridades que devem ser levadas em consideração, ao debater a escassez de documentação de requisitos em metologias ágeis, conforme Mendes et al. [36]. Entretanto, apesar dos problemas levantados, o padrão de documentação ágil agrega diversos benefícios ao processo de desenvolvimento [11], principalmente em relação a aproximação entre os *stakeholders* do projeto e a evolução dos requisitos ao longo do tempo, de maneira dinâmica [29].

Ainda segundo Ernst [20], dados os requisitos **G**, e as suposições de domínio **D**, encontramos a especificação **S**, de tal forma que, juntos, **D** e **S** satisfazem os requisitos **G**. Os requisitos **G** são representados como objetivos (*goals*), que capturam as propriedades desejadas do sistema, incluindo se a satisfação de um objetivo é obrigatória ou opcional com relação a outros objetivos. Os objetivos capturam o que o cliente considera valioso. Os objetivos podem ser satisfeitos pelas especificações **S**, representadas por conjuntos de Tarefas **S** referentes aos comportamentos do futuro sistema. Convêm frisar que as especificidades de **S** são deliberadamente vagas: um conjunto de tarefas pode assumir a forma de delegações, compromissos, serviços, ou código, entre outros. Finalmente, os pressupostos do domínio **D** são condições que se acredita que se mantêm no mundo, bem como afirmações que descrevem como o próprio problema está estruturado.

Neste sentido, quando nos referimos a evolução dos requisitos de maneira dinâmica, nos deparamos com o conceito de *living documentation*, ou *documentação viva*, geralmente detalhada a partir de exemplos, como mostra [2]. Sabe-se que requisitos ágeis são baseados em histórias de usuário, definindo os comportamentos do sistema de maneira simples e sucinta [33], como apresenta a Listagem 1.1.

Listing 1.1: Exemplo de História de Usuário.

```
Como um <tipo de usuario>,
Eu desejo <algum objetivo>
De modo que <algum motivo>.
```

Dessa forma, a abordagem ágil de alguma forma já facilita nas especificações de histórias de usuários saber-se diretamente qual o principal *stakeholder*, qual o objetivo e o motivo que cada história deve atender.

A fim de facilitar a definição e a manutenção de uma documentação viva, surgem algumas estratégias como a abordagem do *Behavior Driven Development* (BDD) [39]. A abordagem BDD é uma derivação do *Test Driven Development* (TDD) [6] e do *Domain Driven Design* (DDD) [21], a qual se baseia na definição de cenários de uso para registro do comportamento esperado do software, possibilitando a automação do cenário de uso para validação deste comportamento [52]. Geralmente, as especificação de cenários BDD são realizadas com apoio da linguagem Gherkin [13], possuindo as palavras-chave necessárias para definição de uma *feature*, seus cenários e seus respectivos passos. Dessa forma, os cenários de uso, na abordagem BDD, são vistos como os requisitos e, ao mesmo tempo, os casos de teste que validam o funcionamento deste requisito, mesclando ambos artefatos em apenas um: uma *feature* BDD. Além disso, a abordagem BDD viabiliza, diretamente, a rastreabilidade entre os requisitos e o código-fonte, conforme demonstrado por Lucassen [32].

De acordo com Lamsweerde [54], na Engenharia de Requisitos Orientada por Objetivos, do inglês *Goal-Oriented Requirements Engineering* (GORE), os requisitos são definidos a partir dos objetivos organizacionais dos envolvidos. Nas fases de elaboração, estruturação, especificação, análise, negociação e documentação, são utilizados os objetivos dos *stakeholders*.

Os modelos de objetivos representam explicitamente requisitos funcionais e não funcionais através da decomposição de objetivos, conforme Kolp et al. [27]. Esta decomposição indica como satisfazer determinado objetivo, indicando a razão pela qual os subobjetivos são necessários. A técnica de modelagem i* [60] é uma das mais relevantes abordagens GORE. Esta técnica fornece uma visão dos atores e suas dependências.

O modelo de requisitos descrito a partir da técnica i* provê uma representação mais completa dos requisitos de um sistema. As dependências entre as funcionalidades ou, mais especificamente, de seus atores organizacionais, que são inexistentes no modelo de *features* BDD, agora são possíveis de serem vistas. Além disso, a leitura desse modelo possibilita a compreensão do contexto a qual uma funcionalidade está inserida ao fornecer uma visão geral das características do sistema [43].

Apoiado nas vantagens de se descrever requisitos utilizando o modelo i*, além de basear-se nas vantagens da especificação dos requisitos usando a técnica BDD no contexto da metodologia ágil e essas carências elencadas ao representar os requisitos do um sistema, torna-se interessante desenvolver uma abordagem capaz de unir um modelo muito eficiente, porém, com limitações de informação, que é a utilização de *features* BDD, para um modelo mais completo, que nos forneça uma visão hierárquica das *features* BDD, demonstrando como suas respectivas tarefas (*tasks*) são capazes de representar os requisitos, por meio do mapeamento das *features* BDD, o modelo i*. E apoiado nesta união nos possibilitar realização do cálculo da $satisfacao$[1] de requisitos.

1.2 Problema

Apesar das várias propostas de gestão de requisitos no processo de desenvolvimento de software, ainda carecem propostas para contornar problemas ainda presentes na gestão de requisitos. Dentre eles temos o estudo de Abad e Ruhe [1] que indicam a aplicação da análise de opções reais para gerenciamento em situações de incerteza, quanto ao contexto do projeto, custo e cronograma. Porém, a forma de gerenciamento proposta não foi testada em um ambiente real de desenvolvimento de software, necessitando assim, de confiabilidade quanto aos seus resultados.

Além disso, conforme Soares et al. [51], dentre as dificuldades no uso de requisitos ágeis encontra-se a falta de informação. Embora o uso de requisitos ágeis possa proporcionar um ganho inicial em termos de tempo durante as atividades de especificação de requisitos, dificuldades como a falta de um requisito detalhado podem causar o desenvolvimento de funcionalidades que não estão alinhadas com as esperadas pelo cliente. Consequentemente, vários indicadores podem emergir: (1) documentação dos requisitos inexistente (ou incompleta), (2) documentação desatualizada por falta de informação ou por volatilidade dos requisitos e (3) documentação de testes carecendo de uma relação com e entre os requisitos. Uma vez que a documentação viva proposta pela abordagem BDD pode explicitar as questões relativas aos três problemas acima, ela ainda carece de prover uma visão global da relação entre os requisitos.

Em particular, ao descrever os requisitos a partir *features* BDD, percebe-se a carência com que essas *features* representam às funcionalidades do sistema, pois não é possível compreender as dependências entre as *features*, além de não ser possível dizer qual o contexto na qual elas estão inseridas dentro de um sistema. Desta forma, faz-se necessário a combinação da abordagem BDD com outra técnica para suprir esta lacuna.

[1]Conceito definido no âmbito deste trabalho como o grau de aceitação dos requisitos expressos por meio das *features* BDD

Por outro lado, a Engenharia de Requisitos Orientada por Objetivos [55] (GORE) provê uma abordagem mais estratégica do processo de desenvolvimento de software em uma perspetiva mais abrangente a partir dos objetivos principais do sistema e seus refinamentos AND/OR. No entanto, abordagens como por exemplo o RE-KOMBINE [20], que tratam a evolução dos requisitos, não provêm uma maneira pragmática de conectar os requisitos técnicos (objetivos) e suas representações mais concretas para uma abordagem ágil.

1.3 Pergunta de Pesquisa

As *features BDD* são bastante fragmentadas e não tem nenhuma relação direta entre si. Assim, torna-se necessário prover uma visão em que seja possível perceber se os objetivos de um sistema de software estão adequadamente atendidos/satisfeitos a partir do conjunto de requisitos especificados nessas *features*.

Dessa forma, acreditamos que as abordagens BDD e GORE têm informações suficientes para prover uma metodologia de mensuração de evolução dos requisitos. Além disso, postulamos, também, que é possível propor uma abordagem para medir o grau de aceitação dos requisitos sem adicionar um alto custo adicional ao processo de desenvolvimento de software, em particular ao processo ágil.

Uma vez que tanto a abordagem BDD, baseada no conceito de documentação viva a partir de histórias de usuário, quanto a abordagem GORE tem como cerne os objetivos do software como requisitos principais, surge então a seguinte pergunta de pesquisa:

> RQ1: É possível saber se o conjunto de *features BDD* que representam o sistema é amplo o suficiente para atender aos seus objetivos de negócio? Se for possível, podemos rastrear diretamente os objetivos do sistema a partir dos seus artefatos de software? Caso essa rastreabilidade seja alcançada, é possível ter uma visão dinâmica e sempre atual dos requisitos alcançáveis por meio de suas *features BDD* e o grau de realização das mesmas?

Assumimos neste trabalho que os requisitos são especificados como *features* no formato Gherkin segundo a abordagem BDD. Aliado a isso utilizamos a abordagem de modelagem de objetivos iStar. Logo unindo essas duas abordagens investigaremos a possibilidade de propor uma abordagem para a gestão do grau de aceitação de requisitos ágeis.

1.4 Objetivo Geral

Devido à dificuldade para gerir a evolução dos requisitos em projetos de desenvolvimento de software, o presente trabalho tem o propósito de investigar formas de medir o grau de aceitação dos requisitos, em projetos de desenvolvimento ágeis. Dessa forma, o atual trabalho teve como foco prover uma metodologia alternativa para o acompanhamento do grau de aceitação dos requisitos. E assim, utilizando recursos da "documentação viva", por meio da técnica do *Behavior-Driven Development* (BDD) e da modelagem *Goal-Oriented Requirements Engineering* (GORE), contribuir na gestão da $satisfacao$ de requisitos em projetos de desenvolvimento de software. Para avaliação da metodologia, o trabalho teve como foco atender ao ecossistema de software do Exército Brasileiro, tendo como referência o projeto de cooperação entre o Centro de Desenvolvimento de Sistemas do Exército Brasileiro e o Departamento de Ciência da Computação da Universidade de Brasília.

1.5 Objetivo Específicos

- Contribuir com o estado da arte sobre a gestão de requisitos.
- Apresentar uma abordagem que evidencie como *Behavior-Driven Development* (BDD) e *Goal-Oriented Requirements Engineering* (GORE) podem contribuir para identificação e medição da $satisfacao$ de requisitos.
- Avaliar a abordagem proposta em um estudo de caso real.
- Permitir que a abordagem proposta possa ser utilizada pela comunidade de software livre e em projetos de desenvolvimento de software ágeis.

1.6 Contribuições da pesquisa

A primeira contribuição deste trabalho consiste na metodologia para cálculo da $satisfacao$ de requisitos. Neste sentido, nossa abordagem visa medir o grau de aceitação dos requisitos, por meio de *features* BDD com foco em identificar a realização com sucesso dos cenários destas respectivas *features* estruturadas e relacionadas por meio de uma árvore de objetivos segundo a abordagem GORE.

Como segunda contribuição deste trabalho, realizamos um estudo de caso para a avaliação da proposta da $satisfacao$ no projeto de desenvolvimento do Sistema de Boletins e Alterações (SISBOL). Desta forma, com relação a este projeto obteve-se 100% dos dados das *features* do projeto. O conjunto de arquivos *.feature* e *.json* foram obtidos a partir

do repositório de código do projeto. Os dados referem-se as *features* implementadas ao longo de 16 *sprints* no período de Outubro de 2016 a Abril de 2018.

Como terceira contribuição deste trabalho, implementamos uma ferramenta que incorpora a metodologia proposta nesta pesquisa. A implementação foi feita a partir da extensão da ferramenta piStar e está disponível em um repositório de código aberto no GitHub[2].

1.7 Organização do Trabalho

Este trabalho está organizado de acordo com o apresentado a seguir, além de conter alguns anexos e apêndices que devem ser consultados para compreensão adequada da pesquisa.

- Capítulo 2: Apresenta o referencial teórico sobre requisitos e as abordagens BDD e GORE. Dessa forma, neste Capítulo são apresentados os conceitos chave para compreensão da pesquisa.
- Capítulo 4: Apresenta a avaliação da proposta levantada nesta pesquisa. Como avaliação, destaca-se a a apresentação da viabilidade da abordagem proposta a partir de análises em diversas configurações de *sprints* do processo de desenvolvimento do SISBOL.
- Capítulo 4: Apresenta a abordagem proposta neste trabalho e as etapas do processo proposto para o gerenciamento da $satisfacao$ de requisitos.
- Capítulo 5: São apresentados os principais trabalhos relacionados a esta pesquisa, situando este trabalho em relação ao estado da arte.
- Capítulo 6: Finalmente, apresentamos as conclusões do nosso trabalho e levantamos os próximos passos vislumbrados para trabalhos futuros.

[2]https://github.com/lealfb/bdd2Goal/

Capítulo 2

Referencial Teórico

Os engenheiros de software, em seus trabalhos, devem procurar a alta qualidade e, ao mesmo tempo, o menor custo possível e o menor prazo para a entrega dos produtos, mesmo que a entrega seja subdividida em fases. Entretanto, o desenvolvimento de software nem sempre segue um caminho suave, pois a condução dos processos tornam-se imprevisíveis, ora por imposição dos recursos de tecnologia disponíveis, treinamento ou mudanças frequentes dos requisitos, ora por imposição do cliente, face às inúmeras frentes que se descortinam durante a construção e a lapidação do software futuro. Nesse contexto, os engenheiros de software estão cientes de que algumas questões técnicas e sócio-técnicas podem comprometer o software e sua evolução. Este capítulo providencia um referencial teórico sobre essa questão no viés das Metodologias Ágeis e suas implicações durante o processo de maturação, assunto relevante para a Engenharia de Software e suas nuances, mostrando o que pensam alguns autores sobre o assunto.

2.1 Desenvolvimento Ágil de Software

Os primeiros movimentos em direção ao Desenvolvimento Ágil ocorreram no ano de 2000, em Oregon, quando alguns engenheiros se reuniram para debater a temática. Mas foi somente em 2001 que, em uma segunda reunião em Utah, é que foi oficializado o movimento, com a publicação do Manifesto Ágil [7]. Com o tempo, os processos foram amadurecendo, surgindo várias metodologias de emprego, como o XP de Kent Beck [5] e o Scrum de Schwaber [47], entre outras.

2.1.1 Conceito

Uma definição para Metodologia Ágil, em toda a sua complexidade e essência, poderia ser resumida em Ericksson et al. [19]:

> "Agilidade significa despir o máximo de peso, comumente associado com metodologia tradicional de desenvolvimento de software, o quanto possível, para promover a rápida resposta às mudanças ambientais, mudanças nos requisitos de usuário e acelerar prazos do projeto."

...e concluindo que "métodos ágeis constituem um conjunto de práticas de desenvolvimento que foram criadas por profissionais experientes..." [3], não obstante, podemos encontrar afirmativas como "...esses métodos podem ser vistos como uma reação ao extenso planejamento tradicional, que enfatiza uma engenharia baseada em racionalidade e em abordagem." [37]. Outras afirmações do tipo "...os problemas são completamente determináveis e que soluções ótimas e previsíveis existem para cada tipo deles." [10], também são plausíveis na literatura.

Hoje, as metodologias ágeis estão bem consolidadas. Uma recente revisão de literatura proposta por Dikert and al. [16], apresenta um panorama bastante promissor, no sentido da adoção dessas metodologias em larga escala industrial. Mesmo assim, embora relatem vários aspectos a favor delas, e como nem todo software pode ser considerado blindado, problemas também foram relatados. Veja o que diz Dikert et al...

> "[...] Nossa análise mostra que houveram estudos bem promissores da utilização de métodos ágeis ao redor do mundo. Embora os estudos identifiquem sérias limitações, como a insustentabilidade do local do cliente dentro do projeto por longos períodos e a dificuldade de introdução de métodos ágeis em grandes e complexos projetos, **os resultados da nossa revisão sugerem que os métodos ágeis podem melhorar a satisfação no trabalho, a produtividade e a satisfação do cliente**."

No Brasil, em pesquisa recente de Mazuco [34], podemos observar a evolução e tendência das Metodologias Ágeis, desde as pesquisas de Dyba, em 2008 [18], passando por Melo, em 2012 [35] até a época mais atual Figura 2.1.

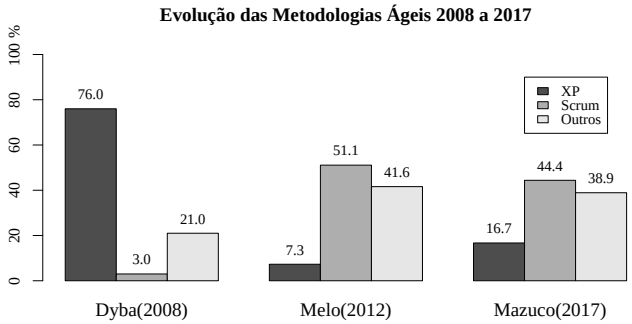


Figura 2.1: Evolução das Metodologias Ágeis entre Dybå, Melo e Mazuco, segundo [34].

E como problemas de desenvolvimento de software estão sempre presentes, incluindo as metodologias ágeis, surge um conceito bem atual que permeia a lide diária na construção, teste e até entrega de produtos sob essa égide: A $satisfacao$ de requisitos. Como é a percepção disto e como isto se apresenta na literatura é o foco deste capítulo.

2.1.2 Desenvolvimento Ágil e a Qualidade

As Metodologias Ágeis surgiram com a promessa da redução de vários problemas com os quais a Engenharia de Software se depara, como a redução de documentação em excesso [4] ou a participação efetiva do cliente junto aos processos de desenvolvimento [28]. A própria detecção desses problemas geraram as práticas denominadas ágeis, hoje bem consolidadas, em cada uma das metodologias que ora se apresentam. Mas, se de um lado as práticas ágeis colaboram para a redução de determinados problemas [31], por outro podem contribuir para a geração de outros problemas, muitas vezes de difícil detecção, o que implicaria em um enorme desafio por parte da equipe ágil em responder às necessidades ou requisitos em constante mudança [53]. Cita-se, por exemplo, a priorização da entrega, onde muitas vezes a qualidade pode ser relegada a segundo plano, o que acaba engrossando as fileiras da falta de qualidade. A grande parte desses problemas nem se quer é conhecida, muito menos acompanhada ao longo do processo de evolução de software ou gerenciada [9], implicando assim em altos custos de manutenção ao longo do ciclo de vida do software.

2.2 Requisitos Ágeis

O processo de elicitação de requisitos é uma atividade crítica para o processo de desenvolvimento de software, possibilitando, ou não, uma entrega de sucesso. A engenharia de requisitos se empenha em identificar, analisar, documentar e validar os requisitos para que o sistema seja desenvolvido [40].

Entretanto, há um desafio no contexto de negócios que faz com que o desenvolvimento do software exija maior atenção ao lidar com requisitos que tendem a evoluir rapidamente, visto que a volatilidade desses requisitos implica no que se quer, de fato, entregar na conclusão de um projeto [11].

As abordagens ágeis de desenvolvimento de software entraram em ascensão a partir da tentativa de maximizar a eficiência nas entregas do software. Os valores do "Manifesto Ágil"[8], descrevem tais abordagens como uma cultura que reúne requisitos durante todo o processo de desenvolvimento, fazendo com que a metodologia dependa da interação com o cliente e atenda às suas mudanças e necessidades. Em termos práticos no fluxo de desenvolvimento, a metodologia ágil faz referência ao *desenvolvimento incremental* [15]. No qual, um conjunto de requisitos é priorizado para um desenvolvimento inicial

e, ao longo do processo de desenvolvimento, novos requisitos e especificações podem ser incluídas de acordo com as interações com o cliente. O *feedback* por parte dos *stakeholders* é fundamental e, com base nisso, os requisitos são ajustados ao longo do tempo. Sendo assim, os demais requisitos são selecionados para a próxima iteração do desenvolvimento, aprimorando as entregas a partir da obtenção de *feedbacks*.

A necessidade de interação entre a equipe de desenvolvimento e os *stakeholders* do projeto levanta uma problemática sobre a estratégia utilizada para definição e registro das funcionalidades do sistema, visto que as técnicas utilizadas por desenvolvedores se mostram bastante obscuras para *stakeholders* que não possuem conhecimento técnico, como destaca Adzic [2].

De acordo com Inayat et al. [24], o conceito de *agile requirements engineering* indica que o processo de elicitação dos requisitos é baseado na maneira "ágil" de planejar e executar atividades relacionadas à Engenharia de Requisitos. Além da comunicação frequente com o cliente e o processo iterativo, destaca-se a utilização do conceito de histórias de usuário, o que facilita e agiliza bastante o processo de elicitação dos requisitos. Segundo Ramesh et al. [42], além dos benefícios já citados, as histórias de usuário possibilitam que o custo e o prazo do projeto sejam estimados, além de facilitar o escalonamento de demandas durante o processo de desenvolvimento.

A Tabela 2.1 apresenta algumas características que merecem destaque na comparação entre a Engenharia de Requisitos tradicional e a Engenharia de Requisitos de acordo com a abordagem ágil [42].

Atividades ER	**Abordagem Tradicional**	**Abordagem Ágil**	**Práticas Ágeis que apoiam a ER**
Elicitação de Requisitos	Definir todos os requisitos no início do projeto	Definição dos requisitos distribuída por todo o processo de desenvolvimento	- ER iterativa -Comunicação cara-a-cara
Análise de requisitos e negociação	Foco na resolução de conflitos	Foco no refinamento, mudança e priorização de requisitos iterativamente	- ER iterativa - Comunicação cara-a-cara - Planejamento constante - Priorização extrema
Documentação dos Requisitos	Documentação formal, contendo requisitos detalhados.	Sem documentação formal e detalhada. Baseado, basicamente, em Histórias de Usuário.	- Comunicação cara-a-cara
Validação dos Requisitos	Análise da consistência e completude do requisito.	Foco em analisar se o requisito contempla a necessidade do cliente.	- Reuniões de Revisão - Comunicação cara-a-cara

Tabela 2.1: Comparação entre abordagens tradicional e ágil em relação ao processo de Engenharia de Requisitos [42].

Com o objetivo de validar as entregas de software de acordo com os requisitos elicitados, a utilização de testes de aceitação é uma prática comum em abordagens ágeis de desenvolvimento, como informa Chow e Cao [11]. A implementação de testes de aceitação pode se tornar uma atividade simples com a utilização de histórias de usuário, podendo ser utilizados para um desenvolvimento guiado por teste de aceitação, ou *Acceptance Test-Driven Development* (ATDD). Além disso, como foi destacado na Tabela 2.1, atividades de interação entre a equipe de desenvolvimento e os clientes são apoiadas

por práticas comuns da Metodologia Ágil. Desse modo, atividades de especificação de histórias de usuário e escrita de testes de aceitação utilizando linguagem natural podem facilitar bastante o processo de Elicitação de Requisitos, como sugere Inayat et al. [24].

Com a percepção dos benefícios de práticas como estas, algumas abordagens, como o *Behavior-Driven Development*, apresentado durante a seção seguinte, são utilizadas como práticas do desenvolvimento ágil.

2.3 Behavior-Driven Development (BDD)

Com o objetivo de maximizar a eficiência dos processos Elicitação de Requisitos e de Verificação e Validação de software, diversas estratégias e metodologias são propostas pela comunidade. Entre elas, vale ressaltar os conceitos de *Acceptance Test Driven Development* (ATDD) e *Test-Driven Development* (TDD), já que a união dos dois conceitos deu fruto à abordagem de *Behavior-Driven Development* (BDD), segundo Solis Et. al [52].

De acordo com Kent Beck [6], na abordagem TDD, os testes do sistema são escritos antes que seu código seja implementado. Com esta abordagem, problemas relacionados à tendência dos testadores são solucionados. Ou seja, como o código ainda não foi escrito, evita-se que o teste seja implementado de maneira enviesada. Entretanto, apesar das reconhecidas qualidades da abordagem TDD, Dan North [38] apresenta alguns problemas ainda presentes na abordagem, principalmente relacionados a linguagem utilizada para escrita dos testes. A maneira com que os testes são escritos tradicionalmente, na abordagem TDD, fazem com que os *stakeholders* do projeto se distanciem da equipe de desenvolvimento e de teste, visto que a linguagem utilizada para a escrita dos testes só é reconhecida por desenvolvedores.

A partir das vantagens e desvantagens observadas na abordagem TDD, surge o conceito de *Acceptance Test Driven Development* (ATTD), aplicando o mesmo fluxo de execução proposto pela abordagem TDD, entretanto utilizando testes de aceitação para guiar o processo [6]. A abordagem ATDD ajuda a equipe de desenvolvimento a transformar requisitos do software em casos de teste, permitindo que o processo de validação do sistema seja realizado com mais facilidade e eficiência. De acordo com Gojko Adzic [2], durante a última década, a comunidade de desenvolvimento tem se esforçado para desenvolver softwares da maneira correta. Entretanto, não basta saber se o software está sendo desenvolvido da maneira correta, deve-se avaliar se o desenvolvimento está gerando o software correto. Para isso, Gojko Adzic [2] destaca as seguintes necessidades:

- Evitar desperdício de tempo durante a especificação do sistema;

- Possuir artefatos confiáveis e de fácil manipulação que definem como o sistema deve se comportar;
- Possuir uma maneira eficiente de checar se o sistema faz o que sua especificação diz;
- Uma maneira de manter a documentação relevante e confiável com o mínimo custo;
- Uma maneira de distribuir as atividades em interações curtas, facilitando a flexibilidade e a capacidade reagir a mudanças.

Englobando as necessidades levantadas anteriormente, Dan North [38] propõe a abordagem de *Behavior-Driven Development*, reunindo as qualidades da Especificação por Exemplo com as qualidades da abordagem de *Test-Driven Development* (TDD). A abordagem BDD utiliza o mesmo ciclo de desenvovimento utilizado pela abordagem TDD, representado pela Figura 2.2, entretanto fazendo uso de uma linguagem ubíquia[1] para definição dos casos de teste, o que evita o distanciamento entre os desenvolvedores e os clientes do projeto, como afirma Solis et. al [52].

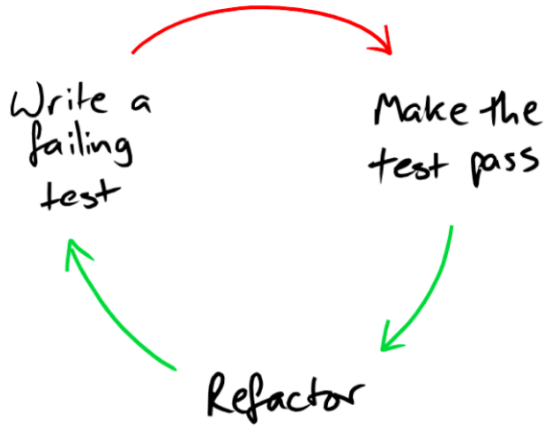


Figura 2.2: Ciclo de desenvolvimento TDD [26].

O uso da abordagem BDD pode ajudar equipes a focar esforços na identificação, compreensão e implementação das funcionalidades do software, segundo John Ferguson [49]. De acordo com Dan North [38], a abordagem foi pensada de tal forma que a escrita dos cenários fosse suficientemente flexível, porém suficientemente estruturada, possibilitando a automação e o fácil entendimento ao mesmo tempo. A estrutura é baseada no seguinte formato:

[1]Linguagem na qual a estrutura da mesma é derivada do domínio de negócio, contendo termos que serão utilizados no contexto real de utilização do sistema para definição do comportamento do sistema. [21]

Dado algum contexto inicial
Quando algum evento ocorrer
Então devo receber o resultado X

Esta estrutura define um cenário na abordagem BDD, porém, de acordo com Solis et. al [52], a abordagem contempla outras informações úteis, como pode-se observar na Figura 2.3. A definição dos cenários de uso do sistema, na abordagem BDD, faz o papel do processo de elicitação de requisitos como conhecemos tradicionalmente. Ou seja, todo o processo de elicitação de requisitos, na abordagem BDD, é realizado a partir da definição de cenários de uso que descrevem o comportamento esperado do sistema, como afirma Santos et. al [17].

Com esta abordagem de especificação de requisitos, temos que os *stakeholders* do projeto se tornam capazes de descrever e documentar os requisitos de seu próprio sistema, participando, ativamente, da equipe de levantamento dos requisitos, como sugere a abordagem ATDD proposta por Gojko Adzic [2]. Além disso, a abordagem ATDD e, consequentemente, a abordagem BDD, sugere uma relação de transparência entre os interessados no produto de software e a equipe de desenvolvimento. Esta transparência se inicia durante o processo de elicitação de requisitos, onde uma linguagem ubíqua é utilizada para que todos os *stakeholders* do projeto possam compreender e até validar os requisitos ainda na fase de escrita dos mesmos. Um conceito importante destacado por Gojko Adzic [2] é a *Living Documentation*, o qual se atenta em manter o contexto atual documentado e atualizado, colaborando na transparência durante o processo de elicitação, desenvolvimento e teste do software. Na abordagem BDD, como os requisitos do sistema são descritos de tal forma que possibilita sua automação durante o processo de Verificação e Validação, o conceito de *Living Documentation* é facilmente coberto pela abordagem.

Geralmente, as especificação de cenários BDD são realizadas com apoio da linguagem *Gherkin*[2], possuindo as palavras-chave necessárias para definição de uma *feature*, seus *scenarios* e seus respectivos *steps*. A Figura 2.3 apresenta um exemplo claro de uma *feature* BDD.

As palavras-chave utilizadas na abordagem BDD são:

- *Feature*: Define uma funcionalidade específica do sistema, contendo um ou mais *scenarios* de uso da *feature*. Geralmente, cada *feature* é especificada em um arquivo com a extensão *.feature*, possibilitando a distinção dos arquivos por parte da ferramenta de processamento utilizada.

- *Background*: Define um cenário inicial que será executado antes de cada *scenario* definido no mesmo arquivo. É composto de um ou mais *Steps*.

[2]https://docs.cucumber.io/gherkin/reference/

```
Feature: Multiple site support
  Only blog owners can post to a blog, except administrators,
  who can post to all blogs.

  Background:
    Given a global administrator named "Greg"
    And a blog named "Greg's anti-tax rants"
    And a customer named "Dr. Bill"
    And a blog named "Expensive Therapy" owned by "Dr. Bill"

  Scenario: Dr. Bill posts to his own blog
    Given I am logged in as Dr. Bill
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."

  Scenario: Dr. Bill tries to post to somebody else's blog, and fails
    Given I am logged in as Dr. Bill
    When I try to post to "Greg's anti-tax rants"
    Then I should see "Hey! That's not your blog!"

  Scenario: Greg posts to a client's blog
    Given I am logged in as Greg
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."
```

Figura 2.3: Exemplo de *Feature* BDD. [25].

- *Scenario*: Define um cenário de uso de sua respectiva *feature*. Deve se localizar no mesmo arquivo de definição da *feature* e é composto de um ou mais *Steps*, definindo o passo-a-passo para execução deste *scenario*.

- *Steps*: Define um passo de execução do sistema. É a unidade básica a abordagem BDD, sendo utilizado para descrever os passos para execução de seu respectivo *scenario*, possibilitando, ainda, a verificação de determinado estado, utilizado para verificar o *output* obtido. Cada *Step* é mapeado, via expressão regular, para um método que implementa o respectivo *step*, fazendo a imersão no código e possibilitando a execução automatizada dos *steps* como métodos do sistema. Os *steps* são baseados nas palvras-chave ***Given***, ***When***, ***Then***, ***And*** e ***But***.

- *Tags*: Informações adicionais podem ser adicionadas a partir da utilização de *Tags*. A partir deste item, pode-se agrupar *features* por usuários e/ou por proximidade de execução, sendo uma informação bastante útil durante o processo de desenvolvimento.

Apesar de possuir poucas palavras-chave, a abordagem BDD é capaz de descrever comportamentos complexos de sistemas de software, fazendo com que *stakeholders* sem conhecimento técnico consigam participar e até escrever *features* BDD, minimizando, consideravelmente, o esforço gasto entre as atividades de Elicitação de Requisitos, Verificação e Validação de software, como afirma Santos et. al [17].

Porém a abordagem BDD, não deixa explícito o quão perto se está de se atingir o objetivo principal de um determinado projeto de software, além disso não demonstra o grau de hierarquia das *features* BDD. Desta forma para suprir esta duas lacunas optamos por fazer uso da técnica da Engenharia de Requisitos Orientada por Objetivos que apresentaremos a seguir.

2.4 Goal-Oriented Requirements Engineering (GORE)

Diferentemente do método tradicional, na Engenharia de Requisitos Orientada a Objetivos ou do inglês, *Goal-Oriented Requirements Engineering* (GORE), de acordo com Lamsweerde et al. [54], os requisitos são elicitados a partir das metas organizacionais dos *stakeholders*, também compreendidas como sendo os objetivos a serem alcançados pelo sistema. Além disso, segundo Lamsweerde et al. [56], na GORE, a análise dos requisitos é fundamentada a partir da investigação sobre as metas da organização, ou seja, em como e quem são os responsáveis por alcançá-las.

Dessa forma, a aplicação da metodologia GORE, segundo Yu et al. [59], é vantajosa para os engenheiros de requisitos por proporcioná-los uma melhor contextualização sobre o problema, obtendo, consequentemente, mais facilidade na compreensão e interpretação dos requisitos do sistema. Dentre outras vantagens, de acordo com Lamsweerde et al. [54], pode-se considerar também a capacidade de as metas fornecerem uma justificativa para os requisitos e, além disso, por se tratar de uma abstração de alto nível, também são capazes de proporcionar aos engenheiros uma melhor orientação nas tomadas de decisão na definição dos requisitos do projeto.

Além do mais, ainda segundo Lamsweerde et al. [54], diversos modelos de requisitos podem ser derivados a partir dos objetivos do sistema, o que faz com que a metodologia seja amplamente aceita pela comunidade. Por essa razão, amparadas na metodologia GORE, diversas técnicas que utilizam as metas dos *stakeholders* já foram desenvolvidas, como: KAOS [14], iStar (ou i*) [60] e NFR [12].

Em nossa proposta faremos o uso do framework iStar que será apresentado a seguir.

2.5 Framework iStar

Com o objetivo de melhorar a compreensão de um sistema de software, a engenharia de requisitos cria modelos que podem ser classificados em dois tipos: de projeto e de requisitos. Enquanto o primeiro busca representar as características do software em termos arquiteturais, priorizando o detalhamento dos componentes, o modelo de requisitos descreve o software a partir do domínio informacional, funcional e comportamental [41].

Proposto por Yu [60], o Framework iStar, também chamado de i* (lê-se: i estrela), é uma abordagem GORE centrada nos *stakeholders* e seus relacionamentos, e permite uma modelagem a partir da utilização de atores que dependem um dos outros para atingirem suas metas. Segundo Carvalho [44], além de ser utilizado na área da engenharia de requisitos, o framework, que é composto por dois modelos - o de dependência estratégica ou SD (do inglês: *Strategic Dependency*) e o de raciocínio estratégico ou SR (do inglês: *Strategic Rationale*) - vem também sendo utilizado por outros campos de pesquisa, como na reengenharia de processos de negócio, de modelagem de software e de desenvolvimento de sistemas orientados a agentes.

A abordagem proporcionada pelo Framework iStar, além de agregar aspectos da modelagem social, trás consigo paradigmas de orientação a metas e a agentes, como também proporciona uma maior atenção nas propriedades intencionais do ambiente e nos seus relacionamentos [23]. Essas intencionalidades são representadas através dos modelos SD e SR.

2.5.1 Modelo SD

O modelo SD fornece uma descrição das intencionalidades do funcionamento organizacional a partir de uma rede de dependências entre atores, conforme pode ser observado na Figura 2.4. A intencionalidade é apresentada por meio de um ator, também chamado de *depender*, que se relaciona com um outro ator, chamado de *dependee*. Nesse modelo, os atores são conectados por um elemento intencional, ou *dependum*. Por fim, a interpretação obtida a partir desses três elementos, é de que o *depender* depende diretamente do *dependee* para satisfazer alguma meta ou objetivo específico, conforme Yu [60].

No exemplo apresentado pela Figura 2.4, o ator "Ator1" depende do ator "Ator2" para atingir determinada meta. O direcionamento do relacionamento é interpretado a partir da orientação do caractere 'D'. Da mesma forma, pode-se inferir, a partir do modelo, que "Ator2" depende de "Ator1" para a realização de uma determinada tarefa. Além do ator, que é representado por um círculo, a Figura 2.4 também apresenta os 4 tipos de *dependums*: meta, meta-soft, tarefa e recurso.

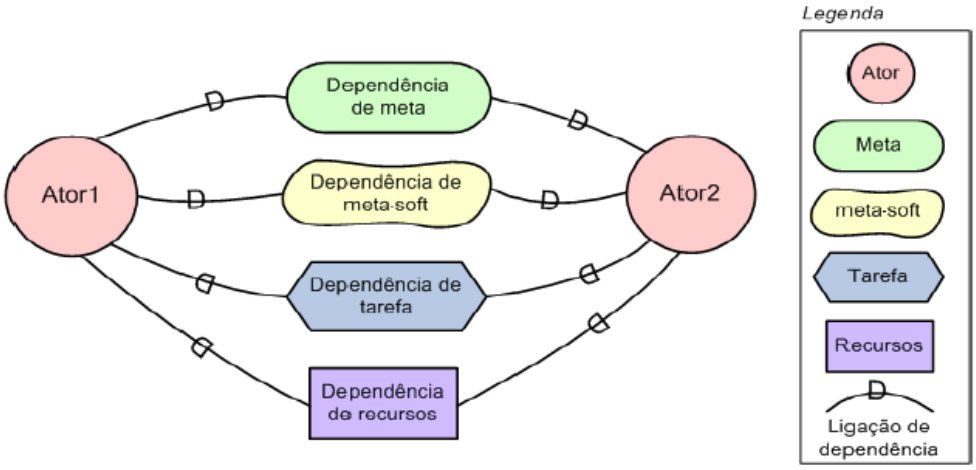


Figura 2.4: Representação da notação do modelo SD. Retirado de Silva [48].

2.5.2 Modelo SR

Diferentemente do modelo SD que descreve os relacionamentos externos, ou seja, as dependências entre os atores por meio de elementos intencionais, o modelo SR apresenta a estrutura interna dos atores, buscando descrever como eles se comportam para atender determinada demanda na organização, conforme Yu [60].

Com base na ilustração do modelo SR da Figura 2.5, pode-se observar uma expansão do ator "Ator2", delimitado por um círculo pontilhado. Nesse modelo, a meta principal do ator expandido está sendo refinada a partir de tarefas. A execução das tarefas "Tarefa1" e "Tarefa2" são alternativas para a realização da meta, sendo que especificamente a primeira apresenta decomposição em outras duas subtarefas. Um elemento de tarefa pode ser vinculado a outras tarefas por links de decomposição, podendo ser disjuntivos (AND) ou conjuntivos (OR). Nos links AND, o a tarefa pai é satisfeita se todos os descendentes estiverem satisfeitos, enquanto na ocorrência do link OR, para que a tarefa pai seja realizada é necessário que no mínimo uma das tarefas filhas sejam satisfeitas. Pode-se interpretar também, ainda a respeito do modelo da Figura 2.5, que a tarefa "Subtarefa1" contribui negativamente para uma meta-soft, enquanto "Tarefa2" contribui de maneira positiva.

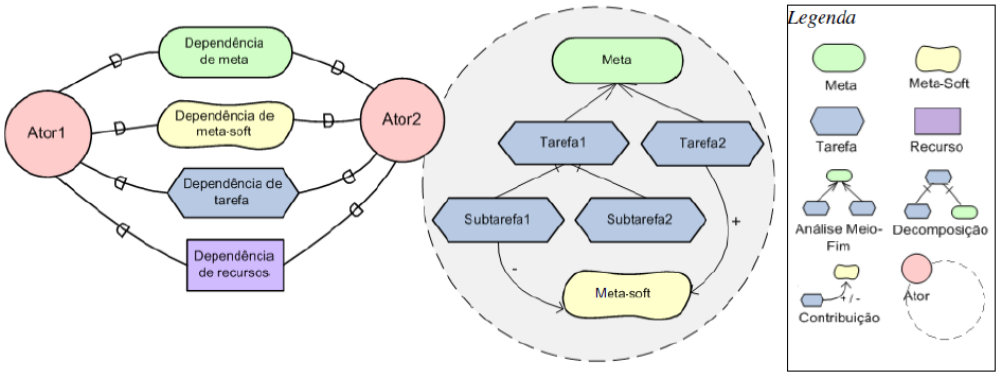


Figura 2.5: Representação da notação do modelo SR. Retirado de Silva [48].

Capítulo 3

Abordagem BDD-GORE

Nossa abordagem visa medir a $satisfacao$[1] em relação atendimento aos requisitos a partir da aferição das features disponíveis no projeto, com foco em satisfazer os objetivos do negócio, representados por meio do Modelo de Objetivos. Nesse Capítulo, apresentamos os passos da abordagem BDD-GORE e em seguida, apresentamos a ferramenta que automatiza os passos da abordagem proposta.

3.1 As Atividades da Abordagem BDD-GORE

A Figura 3.1 apresenta uma visão global da abordagem proposta neste trabalho. Vale ressaltar que a abordagem proposta possui a premissa básica de que o processo de desenvolvimento é baseado na utilização da abordagem BDD e do Modelo de Objetivos.

A abordagem **BDD-GORE** consiste nas seguintes atividades:

1. Mapeamento *features* BDD e Modelo de Objetivos (Seção 3.1.1).
2. Medição do grau de $satisfacao$(Seção 3.1.2).
3. Gestão da $satisfacao$ dos requisitos (Seção 3.1.3).
4. Acompanhamento do grau de $satisfacao$ (Seção 3.1.4).

Nas demais subseções, detalhamos tais atividades da abordagem proposta.

3.1.1 Mapeamento Features BDD e o Modelo de Objetivos

O mapeamento entre os artefatos BDD e os artefatos do modelo de objetivos é feito de acordo com o apresentado na Figura 3.2. A partir desta Figura é possível observar que

[1]Definida neste trabalho como: grau de aceitação dos requisitos expressos por meio das *features* BDD

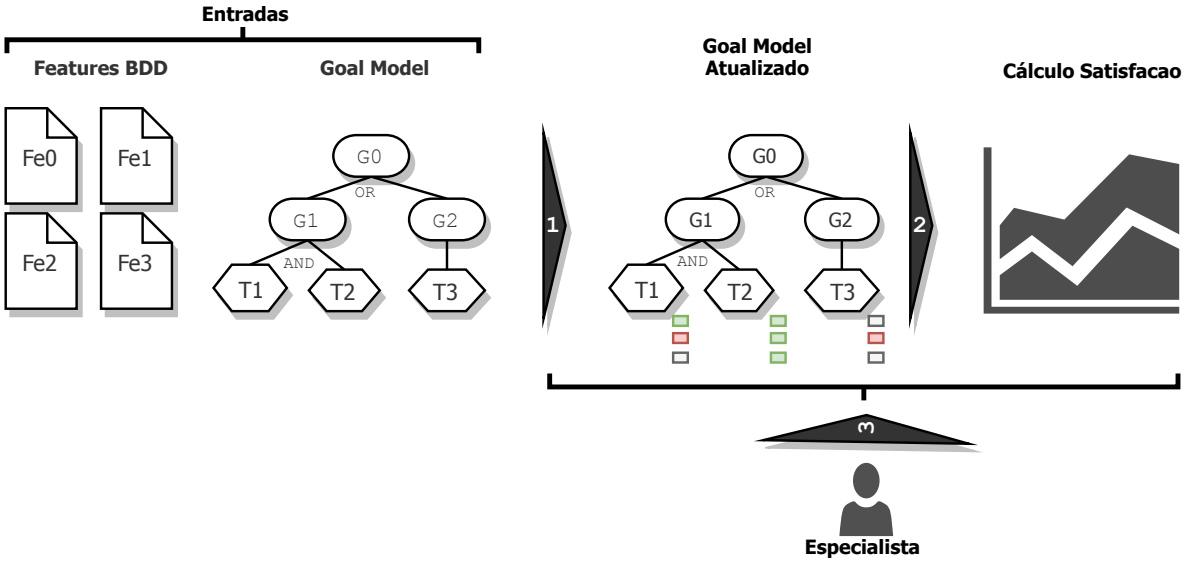


Figura 3.1: Visão geral da abordagem.

cada *feature* BDD é relacionada diretamente com uma *task* do modelo de objetivos. Com o objetivo de facilitar a compreensão da abordagem, a representação do modelo de objetivos foi simplificada em duas entidades: **Goal** e **Task**.

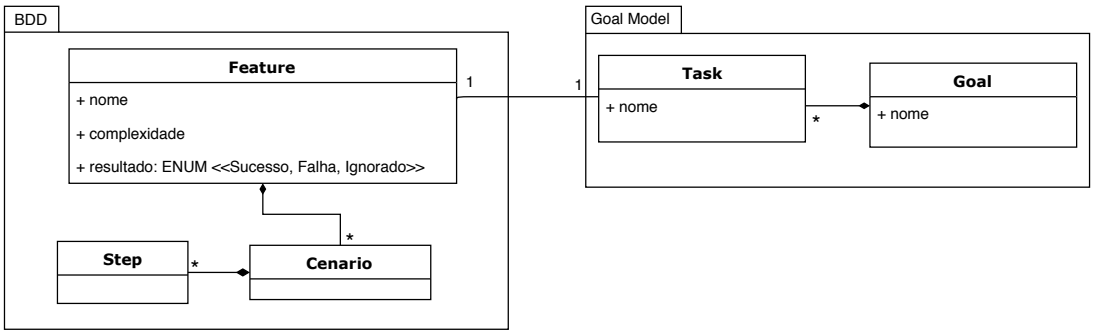


Figura 3.2: Relacionamento semântico entre BDD e Goal Model.

A abordagem BDD é composta de três entidades principais: *Feature*, *Cenário* e *Step*, sendo que uma *Feature* possui um ou muitos *Cenários*, e que um *Cenário* possui um ou muitos *Steps*. Dessa forma, o resultado de uma *Feature* depende do resultado de todos os *Cenários* que a compõem, e o resultado de cada *Cenário* depende do resultado de cada um dos *Steps* que compõem este *Cenário*.

O processo de mapeamento entre as entidades é realizado com base no nome das *features*. Vale ressaltar que nossa abordagem assume que tanto as *features* BDD quanto o modelo de objetivos são previamente especificados. Ou seja, dado que ambos artefatos existem, tanto o modelo de objetivos quanto os artefatos BDD, basta relacionar as *features* e as *tasks* com base no nome de ambas entidades. Esta atividade deve ser realizada com ajuda de um especialista. A partir do mapeamento entre as entidades, as *tasks* têm acesso à informação de complexidade da *feature* relacionada e o resultado da execução da mesma (ignorado, falha ou sucesso). Dessa forma, para a satisfação do objetivo raiz é necessário que todas as suas *tasks* (em refinamentos AND), ou pelo menos uma de suas *tasks* (em refinamentos OR), sejam realizadas com sucesso por meio de suas respectivas *features*.

O próximo passo da abordagem BDD-GORE consiste no cálculo da $satisfacao$, que será apresentado a seguir.

3.1.2 Medição do Grau de Satisfação

Em nossa abordagem, a $insatisfacao$[1] com relação a completude dos requisitos é uma métrica que tem relação inversamente proporcional à satisfabilidade das *features* entregues com sucesso no projeto. Ou seja, quanto maior a insatisfação, menor a taxa de aceitação do *stakeholder* quanto ao requisito entregue e vice-versa. Baseado nessa relação, buscou-se a definição de uma estratégia para medição e acompanhamento da satisfação, que será apresentada a seguir.

Primeiramente, uma vez que cada passo, cada cenário e cada *feature* tem cada um seu próprio *status*, i.e. *skipped*, *failed* ou *successful*, é importante levar esse *status* em consideração para o cálculo da satisfação. Dessa forma, atribuímos o resultado de cada cenário da *feature* com um **peso** para representar o status da execução *feature*, conforme segue:

- *Skipped*: Cenário BDD está definido mas ainda não teve todos seus passos implementados. Neste caso, não é possível, ainda, validar o cenário em questão. Este *status* para o cálculo da satisfação possuirá **peso 0**.
- *Failed*: Cenário BDD definido e com passos implementados. Entretanto, a execução deste cenário apresenta uma falha, que deverá ser corrigida pela equipe de desenvolvimento. Este *status* para o cálculo da satisfação possuirá **peso 1**.
- *Successful*: Cenário BDD definido e com passos implementados. Execução correta do cenário, ou seja, sem a ocorrência de uma falha durante a execução do mesmo. Este *status* para o cálculo da satisfação possuirá **peso 2**.

[1] Grau de **não** atendimento dos requisitos no cumprimento aos objetivos de negócio utilizando a abordagem do *Behavior-Driven Development*

Levando em consideração estes *status* possíveis para cada cenário BDD, podemos definir qual seria a distância entre situação atual e a situação ideal do projeto.

Portanto, o cálculo do **Estado Atual** de uma *feature*, a partir do *status* de seus cenários, pode ser obtido a partir da Equação 3.1:

$$estadoAtual = \left(\sum_{i=1}^{nc} p(c_i) \right) \times c(f) \tag{3.1}$$

Onde,

- nc: número de cenários da respectiva *feature*;
- $c(f)$: representa a complexidade da *feature* f;
- $p(c_i)$: representa o valor do peso associado ao cenário i analisado
  i.e.: *skipped*(0), *fail*(1), *success*(2).

Assim sendo, o **Estado Ideal** pode ser calculado de acordo com a Equação 3.1, onde $p(c_i)$ possui valor 2 em todos os cenários envolvidos. Por outro lado, para o cálculo do estado atual, $p(c_i)$ representa o valor do *status* de cada cenário analisado.

Dessa forma, a partir da relação entre o $estadoAtual$ e o $estadoIdeal$ podemos definir o grau de $satisfacao$ da *feature*, como apresentado na Equação 3.2.

$$satisfacao = \frac{estadoAtual}{estadoIdeal} \tag{3.2}$$

Neste sentido, $satisfacao$ representa o grau de aceitação do produto desenvolvido num intervalo que varia entre 0 e 1. Como derivação da taxa de aceitação do produto de software, podemos levantar a $insatisfacao$ relacionada.

A $insatisfacao$ pode então ser analisada de maneira local ou global. A $insatisfacao$ Local (INS Local) faz referência ao grau de aceitação presente em cada uma das *features* envolvidas. Sabemos que a $insatisfacao$ é inversamente proporcional a taxa de aceitação do produto entregue. Assim sendo, a $insatisfacao$ local pode ser obtida a partir da Equação 3.3.

$$INS\_local(f_i) = 1 - satisfacao \tag{3.3}$$

A $insatisfacao$ Global (INS Global) faz referência ao grau de não aceitação no cumprimento aos objetivo definidos levando em conta todo o processo de desenvolvimento. A INS Global pode ser definida a partir da Equação 3.4.

$$INS\_global = 1 - \frac{\sum_{i=1}^{nf} estadoAtual_i}{\sum_{i=1}^{nf} estadoIdeal_i} \tag{3.4}$$

Onde,

- i: representa cada *feature* mapeada no modelo de objetivos;
- nf: representa o número total de *features*;
- $estadoAtual_i$: representa estado atual *feature* i;
- $estadoIdeal_i$: representa estado ideal *feature* i;

3.1.3 Gestão da Satisfação dos Requisitos

A partir do cálculo da $satisfacao$ realizado na etapa anterior da nossa metodologia, a etapa final consiste em prover informação para dar suporte à gestão do grau de aceitação com relação aos requisitos. Para tanto, o modelo de objetivos torna-se uma estrutura fundamental para essa gestão, uma vez que tal modelo provê uma visão global do sistema sob a perspectiva estratégica do sistema a ser entregue.

Dessa forma, se houver algum ramo da árvore de objetivos que não tiverem suas tarefas ainda realizadas com sucesso, o gestor pode avaliar a forma mais adequada de cumprir o requisito conforme a importância que tal ramo da árvore representa para o modelo de negócios. Se, por exemplo, tal situação acontecer em um refinamento OR, e houver algum outro irmão que tenha suas tarefas realizadas com sucesso, isso traz ao gestor a informação de que provavelmente o foco no cumprimento desse objetivo é de baixa prioridade, caso necessário. No entanto, se em um ramo com refinamento AND houver tarefas que não tiverem sido realizadas ainda, o cumprimento do requisito inevitavelmente precisa ser realizado, uma vez que não há alternativas para a realização do objetivo raiz do ramo em questão, como apresentado na Figura 3.3.

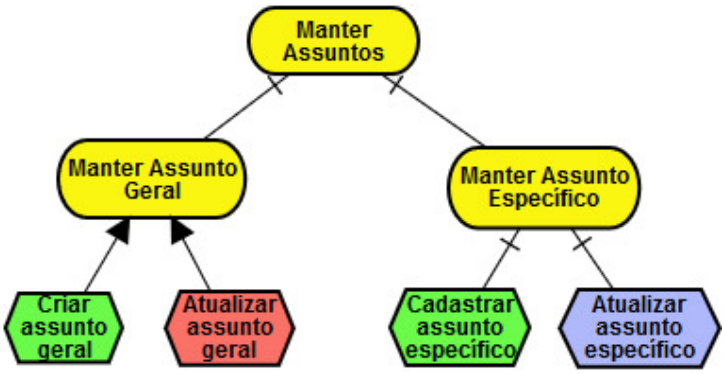


Figura 3.3: Modelo de Objetivos com refinamentos AND/OR.

Na Figura 3.3 temos um objetivo principal *Manter Assuntos* que precisa ser satisfeito. Por sua vez, esse objetivo é refinado em dois objetivos secundários *Manter Assunto Geral* e

Manter Assunto Específico. A realização destes objetivos são obrigatórios para a realização do objetivo raiz. O objetivo *Manter Assunto Geral* configura um refinamento OR com duas sub-tarefas: *Criar assunto geral* com *status* de sucesso e *Atualizar assunto geral* com *status* de falha.

Complementarmente, o objetivo *Manter Assunto Específico* configura um refinamento AND também com duas sub-tarefas: *Cadastrar assunto específico* com *status* de sucesso e *Atualizar assunto específico* com *status* pendente.

A semântica da cor verde representa a tarefa que teve uma *feature* associada com todos os cenários com êxito. A cor vermelha representa a tarefa que teve a *feature* implementada mas apresentou falha e a cor azul denota a tarefa que teve associada uma *feature* não implementada.

Para apoiar a adoção da abordagem proposta neste trabalho foi implementada uma ferramenta de suporte, vale ressaltar que a coloração é realizada de forma automática no momento da associação da *feature* à tarefa no modelo de objetivos. a cor é configurada com base no resultado de execução da *feature* BDD. Apresentaremos a ferramenta na Seção 3.2.

3.1.4 Acompanhamento do Grau de Satisfação

Objetivando ilustrar a viabilidade da utilização desta abordagem para acompanhamento da $satisfacao$ dos requisitos, foram feitas simulações sobre os dados baseados no conjunto de *features* do SISBOL. Este dados são apresentados a seguir em três configurações distintas. A primeira configuração, apresentada na Tabela 3.1, representa um processo de desenvolvimento com sucesso absoluto, ou seja, com todas as *features* implementadas com sucesso na *sprint* correta (sem nenhuma $insatisfacao$). A Tabela apresenta um exemplo de como os dados são registrados, entretanto, por motivo de espaço, o conjunto completo de *sprints* desta configuração podem ser observadas no Apêndice E.

O modelo de objetivos que representa o processo de desenvolvimento destacado na Tabela 3.1 possibilita a análise da evolução da $satisfacao$ ao longo do processo de desenvolvimento. A semântica de cada uma das colunas da tabela é descrita a seguir - Co: complexidade da *feature*, Sk: quantidade de cenários com situação *Skipped*, Fa: quantidade de cenários com situação *Fail*, Su: quantidade de cenários com situação *Sucess*, Tc: total de cenários da *feature*, Sa: situação atual da *feature*, Si: situação ideal da *feature*, Sat: grau de satisfação. Complementarmente a evolução completa é apresentada no Apêndice C.

Um processo de desenvolvimento de sucesso absoluto gera a curvas de $satisfacao$ apresentada na Figura 3.4 (a). Observa-se que a intercessão entre as curvas de $satisfacao$ e $insatisfacao$ Global ocorre aproximadamente no meio do projeto, entre as *sprints* oito e nove. Esse comportamento demonstra a corretude da técnica, já que em um projeto de software de sucesso, o meio

Tabela 3.1: Exemplo de sucesso SISBOL (primeiras 2 sprints).

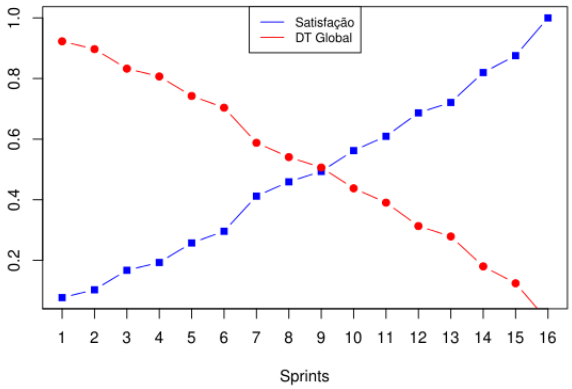
Sprint/Features		**Co**	**Cenários**				**Evolução**			
			Sk	**Fa**	**Su**	**Tc**	**Sa**	**Si**	**Sat**	**1-Sat**
1	Criar assunto especifico	1	0	0	2	2	4	4	1	0
	Criar assunto geral	1	0	0	5	5	10	10	1	0
	Criar posto/graduação	1	0	0	3	3	6	6	1	0
	Criar tipo de documento	1	0	0	2	2	4	4	1	0
	Criar qualificação militar	1	0	0	6	6	12	12	1	0
2	Visualizar assunto especifico	1	0	0	2	2	4	4	1	0
	Visualizar assunto geral	1	0	0	1	1	2	2	1	0
	Visualizar postos/graduações	1	0	0	1	1	2	2	1	0
	Visualizar tipos de documento	1	0	0	1	1	2	2	1	0
	Visualizar qualificações militar	1	0	0	1	1	2	2	1	0

Tabela 3.2: Processo com algumas falhas e *skippeds*.

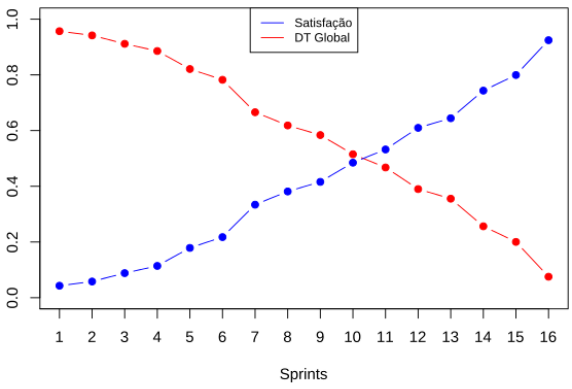
Sprint/Features		**Co**	**Cenários**				**Evolução**			
			Sk	**Fa**	**Su**	**Tc**	**Sa**	**Si**	**Sat**	**1-Sat**
1	Criar assunto especifico	1	0	1	1	2	3	4	75	0.25
	Criar assunto geral	1	0	3	2	5	7	10	70	0.3
	Criar posto/graduação	1	0	1	2	3	5	6	83.33	0.16
	Criar tipo de documento	1	1	1	0	2	1	4	25	0.75
	Criar qualificação militar	1	3	2	1	6	4	12	33.33	0.66
2	Visualizar assunto especifico	1	0	1	1	2	3	4	75	0.25
	Visualizar assunto geral	1	0	1	0	1	1	2	50	0.5
	Visualizar postos/graduações	1	0	1	0	1	1	2	50	0.5
	Visualizar tipos documento	1	1	0	0	1	0	2	0	1
	Visualizar qualificações militar	1	0	0	1	1	2	2	100	0
3	Excluir assunto especifico	1	0	1	1	2	3	4	75	0.25
	Atualizar assunto especifico	1	1	1	0	2	1	4	25	0.75
	Excluir assunto geral	1	1	1	0	2	1	4	25	0.75
	Atualizar assunto geral	1	2	2	2	6	6	12	50	0.5
	Excluir posto/graduação	1	0	1	0	1	1	2	50	0.5
	Atualizar posto/graduação	1	1	1	0	2	1	4	25	0.75

do projeto tende a levar a metade do produto de software concluído, ou próximo disso. Metade do produto de software concluído significa que metade da satisfação dos requisitos especificados nas *features* já foi alcançada.

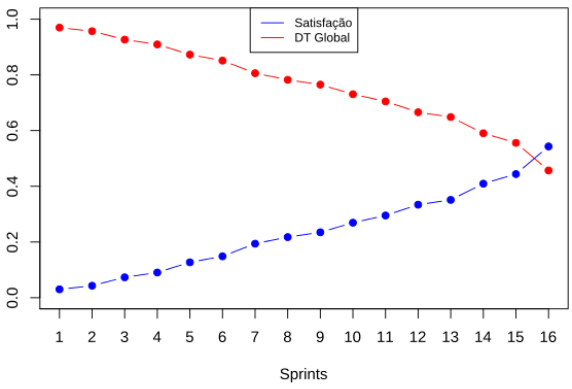
Percebe-se que em tal simulação, o grau de aceitação aos requisitos foi 100% ao final do projeto, sem que houvesse crescimento da $insatisfacao$ de forma exponencial dos requisitos. Em outras palavras, o projeto mostrou-se relativamente bem administrado e com custos dentro do planejado inicialmente pelo projeto no que tange aos requisitos almejados.



(a) Sucesso



(b) Falha no começo e sucesso no final



(c) Falha

Figura 3.4: Comparação entre hipotéticos processos de desenvolvimento.

A segunda configuração, representada pela Tabela 3.2, representa um processo de desenvolvimento com grande dificuldade no início do projeto e acentuada recuperação no fim. Na evolução da $insatisfacao$ para a segunda simulação apresentada na Figura 3.4 (b) observa-se que, como houve a presença de uma $insatisfacao$ mais acentuada no início do projeto, p.ex. o processo de desenvolvimento atrasado, as curvas de $satisfacao$ e $insatisfacao$ se cruzam após a metade do processo de desenvolvimento. Em comparação com a Figura 3.4 (a), a segunda

configuração obteve um atraso de quase dois *sprints* na intercessão entre as curvas de $satisfacao$ e $insatisfacao$.

Já a terceira configuração representa um processo de desenvolvimento de insucesso, ou seja, com $insatisfacao$ do início ao fim do projeto. A Figura 3.4 (c) apresenta o comportamento da $insatisfacao$ ao longo do projeto representado pelos dados disponíveis no Apêndice E.

O modelo de objetivos que representa a evolução da $insatisfacao$ ao longo do processo de desenvolvimento nas configurações 1 e 2 segue a mesma metodologia do apresentado no Apêndice C.

Dessa forma, percebe-se que a compensação da $insatisfacao$ dos requisitos foi melhor realizada na primeira simulação. Enquanto que a compensação da $insatisfacao$ dos requisitos na segunda e na terceira configuração tiveram um certo déficit na entrega, onde a segunda obteve próximo a 100% de satisfação enquanto que a terceira obteve em torno de 60% de satisfação ao final do projeto. Mais ainda, a terceira configuração deixa um déficit de quase 40% a ser paga. Em outras palavras, o terceiro projeto mostrou-se mal administrado e com custos fora do planejado inicialmente pelo projeto no que tange aos requisitos almejados.

3.2 A Ferramenta BDD2Goal

Para melhor entendimento do processo de mapeamento, apresentamos um diagrama de atividades na Figura 3.5 que demonstra o fluxo do mapeamento das *features* para o modelo de objetivos.

O diagrama ajuda no entendimento da relação entre a ferramenta *Serenity* que gerará os relatórios com o resultado da execução dos critérios de aceitação das respectivas *features* e a ferramenta **BDD2Goal**. Além disso, demonstra o passo em que é realizado o mapeamento das *features* para o modelo *i**.

Primeiramente, a ferramenta BDD2Goal solicita ao usuário o arquivo de entrada, exportado pelo *Serenity*, no formato *.json*. Este arquivo contêm o resultado da execução das *features* com a situação da execução de cada cenário e com seus respectivos passos apresentando sua situação: *"Success"*, *"Fail"* ou *"Skipped"*. Caso o usuário não tenha esse arquivo, ele terá que obter por meio da ferramenta *Serenity*, caso ele já possua o arquivo, ele deverá fazer o *upload* na ferramenta BDD2Goal. A Figura 3.6 demonstra como é realizada esta etapa.

Após realizado o *upload* dos arquivos *.json* na ferramenta deverá ser carregado o Modelo de Objetivos do projeto. Vale ressaltar que caso o usuário ainda não tenha o arquivo com o Modelo de Objetivos ele poderá ser construído "online" neste momento. A Figura 3.7 demonstra como é realizada esta etapa.

Após carregado o Modelo de Objetivos o usuário deverá selecionar a *task* que ele deseja realizar a associação com a *feature* BDD. A Figura 3.8 demonstra esta etapa.

Após realizado a leitura do Modelo de Objetivos deverá ser realizado mapeamento da *feature* com a *task* do Modelo de Objetivos.

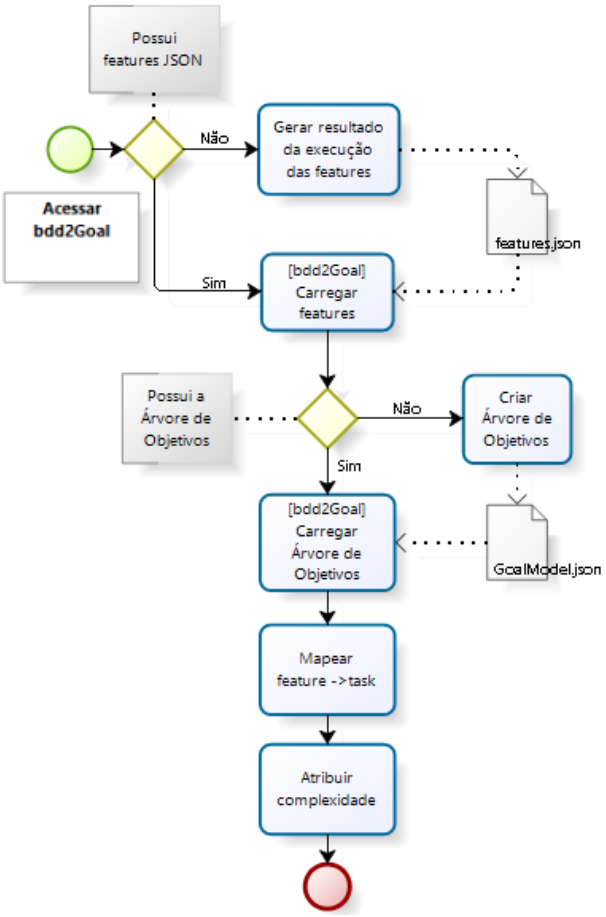


Figura 3.5: Atividades para o mapeamento das *features* para o modelo de objetivos.

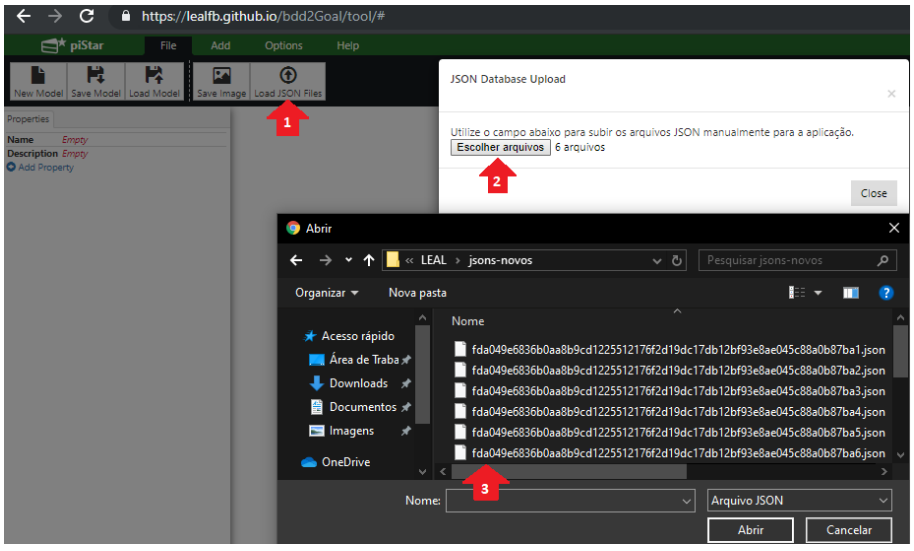


Figura 3.6: Passo 1 - Leitura dos *jsons* que representam as *features* BDD.

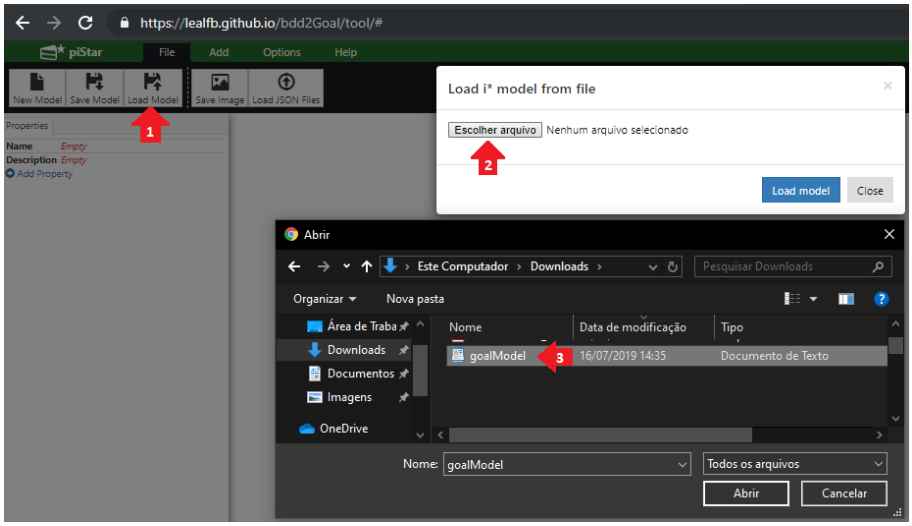


Figura 3.7: Passo 2 - Leitura do Modelo de Objetivos do projeto.

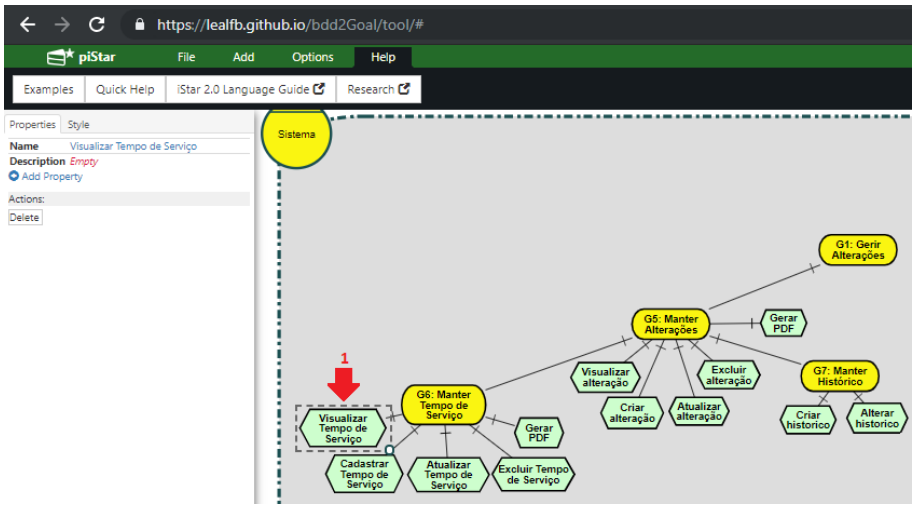


Figura 3.8: Passo 3 - Seleção da *task* no Modelo de Objetivos .

No momento da associação da *feature* a *task* a cor é configurada automaticamente a *task* do Modelo de Objetivos de acordo com o resultado de execução da *feature* carregado via *json*. A *task* pode ser configurada automaticamente com uma das três cores: verde, vermelho ou azul.

Em seguida, há a possibilidade de geração de um arquivo de saída no formato *.json* com a árvore de objetivos já mapeada com as *features* BDD e com sua respectiva complexidade. Este arquivo gerado no formato *.json* pode ser importado pela ferramenta web BDD2Goal que é uma extensão da ferramenta *piStarTool*, desta forma o modelo de objetivos reflete de forma fidedigna a informação provida por meio das *features* BDD. Os passos são exemplificados na Figura 3.8, Figura 3.9 e Figura 3.10

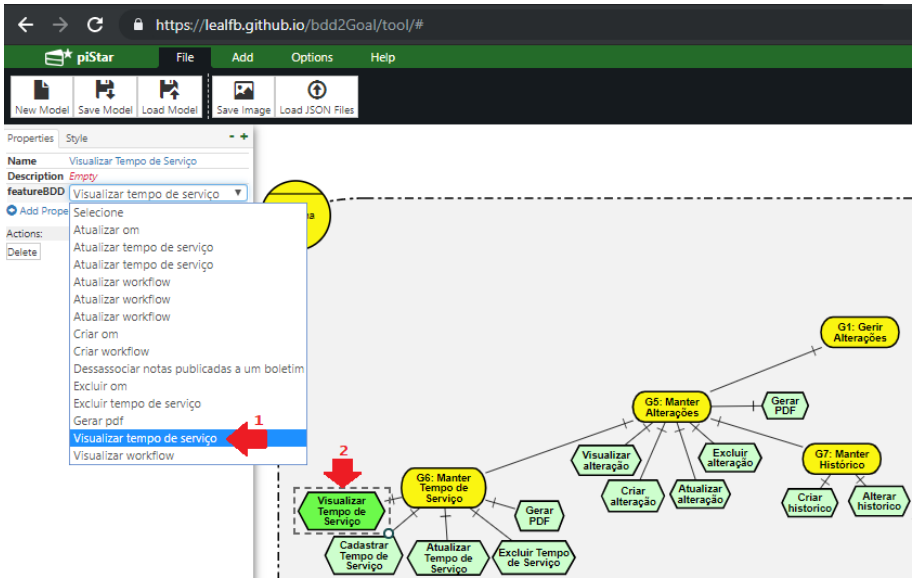


Figura 3.9: Passo 4 - Associação da *feature* a *task* no Modelo de Objetivos.

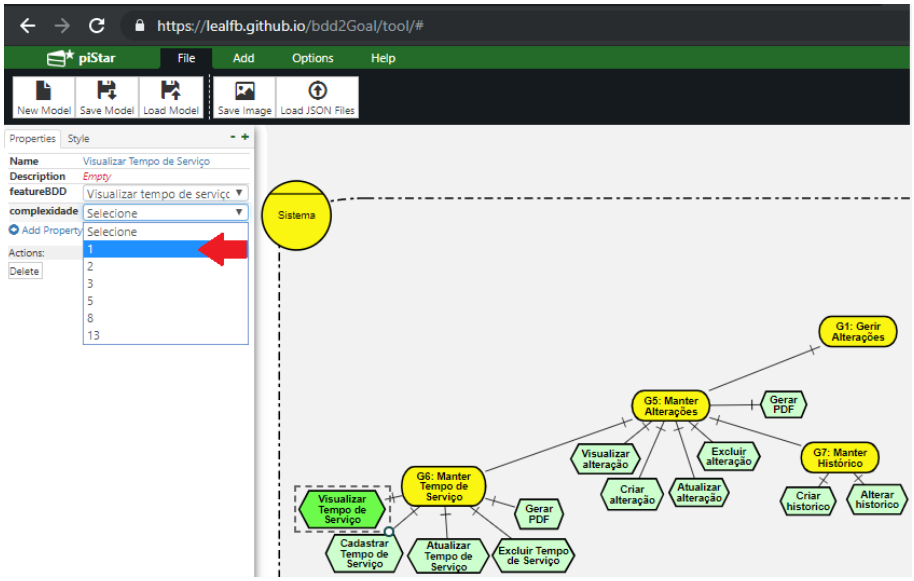


Figura 3.10: Passo 5 - Associação da complexidade a *task* no Modelo de Objetivos.

3.3 Verificação da realização do objetivo

A satisfação do objetivo principal do negócio (raiz) é realizada através do algoritmo de verificação do cumprimento de objetivos, onde alavancamos a atingibilidade do objetivo numa árvore de objetivos (GM), acrescentando a perspectiva das *features* BDD para as *tasks* do modelo. Desta forma, as decisões de adaptação consideram a exequibilidade em que a *feature* BDD pode ser explicitamente associada a *task* e o seu efeito no cumprimento de um determinado objetivo.

No Algoritmo 1 avaliamos a atingibilidade dos objetivos do sistema levando em consideração

o resultado de execução das *features* BDD. O algoritmo tem como informação de entrada um modelo de objetivos (GM) com suas respectivas *features* BDD associadas. O algoritmo é recursivo, baseado no fato de que o GM é um modelo estruturado em árvore e que cada refinamento pode ser visto como um nó de árvore. O algoritmo considera o nó raiz da GM (linha 1).

Algorithm 1: isExequivel(GoalModel gm)

```
   Input: GoalModel gm
   Result: Boolean
 1 Function isExequivel()
 2 |  var result ← false;
 3 |  gm.result ← true;
 4 |  children ← gm.getChildren();
 5 |  for node in children do
 6 |  |  if (node.type = "Goal") then
 7 |  |  |  result ← isExequivel(node);
 8 |  |  end
 9 |  |  else
10 |  |  |  result ← getResultTask(node);
11 |  |  end
12 |  |  if (node.relation = "AND") then
13 |  |  |  gm.result ← gm.result && result;
14 |  |  end
15 |  |  if (node.relation = "OR") then
16 |  |  |  gm.result ← gm.result || result;
17 |  |  end
18 |  end
19 |  return gm.result
```

3.4 Análise Detalhada da Estrutura de Verificação

Basicamente, a função tem como entrada um objeto do tipo *Goal Model*. Inicialmente, iniciamos uma variável para armazenar o resultado da iteração com valor falso (linha 2). Como utilizaremos a soma de valores lógicos para definir a condição do objeto ser ou não ser exequível, o valor inicial para a propriedade de resultado do *Goal Model* deve ser definido como *boolean* "verdadeiro" (linha 3). Uma vez que a soma de valores lógicos dos seus objetos adjacentes do tipo *Task* é feita com operador do tipo *AND*, um *Goal Model* apenas é exequível quando todas suas *Tasks* forem executadas com sucesso. Qualquer resultado diferente disso, é atribuído automaticamente o valor "falso" e toda a soma passa a resultar em "falso". Em seguida buscamos os objetos subjacentes do *Goal* (linha 4), para então realizar uma iteração em todos esses objetos ou nós filhos de *Goal*. Durante essa iteração, é necessário verificar se o objeto subjacente analisado é um objeto do tipo *Goal* (linha 6). Caso seja, a função é chamada recursivamente, enviando o próprio objeto como parâmetro (linha 7), para assim então se tornar possível fazer a leitura de todos os atributos de todos os objetos do modelo. Caso o objeto não seja do tipo *Goal*, é feita a busca do resultado de execução da *Task* (linha 10). Esse resultado é armazenado na variável que foi uma vez inicializada no começo da função do algoritmo (linha 2). Então, dando continuidade à execução do algoritmo, onde é executada a soma do valor final de execução da *Task* ao total já somado logicamente à *Goal* (linhas 13 e 16). Essa atribuição é feita condicionalmente dependendo do tipo de ligação montada entre os objetos *Goal* e Task. Para isso é feita uma verificação ao tipo *AND* (linha 12), caso contrário, é do tipo *OR* (linha 15). Finalizada a iteração dos objetos, a função retorna o resultado final armazenado no *Goal Model*. Lembrando que cada a iteração que esteja em recursividade, o mesmo valor é reatribuído nas outras iterações (linha 7).

3.4.1 Configuração do experimento

O estudo consistiu na avaliação da metodologia do estudo de caso SISBOL. Utilizamos o modelo de objetivos fornecido na Figura 4.1. Modelamos os objetivos e suas tarefas que representavam as *features*. A avaliação foi baseada em um protótipo implementado em *javascript*. Os experimentos foram realizados em uma máquina Intel(R) Core i5 2.8 GHz com 8Gb memória RAM DDR3.

3.4.2 Resultados

O algoritmo é eficiente com relação ao tempo de execução? Avaliamos o tempo de execução do algoritmo para o *Goal Model* do SISBOL. Os resultados mostraram que o algoritmo levou no máximo de 40 milissegundos para ser executado em todos os nós do modelo. Portanto, o algoritmo pode ser considerado bastante eficiente para a configuração analisada. Formalmente falando, o algoritmo pode ser provado como complexidade linear de tempo O(n), onde n é o número de nós no *Goal Model*. A maior complexidade do algoritmo está na execução de cada

node in children (linhas 5-18). Se o tipo do nó for do tipo *Goal*, o algoritmo faz uma chamada recursiva a si. Se o nó for do tipo *task* a variável *result* armazena o resultado da *task*.

O algoritmo permite testar e explicar a alcançabilidade do objetivo baseado nas *features* BDD? Realizamos simulação com 3 configurações de projeto, cada uma contendo um total de 82 *tasks* distintas. As configurações do projeto com *features* com estado de sucesso é demonstrada no Apêndice C e *features* com estado de pendente ou falha no Apêndice D. Ter a raiz alcançada para o conjunto de *features* significa que o objetivo raiz do *Goal Model* do SISBOL é atingido e, portanto, considerando as *features* BDD o sistema é capaz de atingir seus objetivos. Das 82 *features* modeladas, em nenhuma das configurações tiveram os objetivos do SISBOL alcançados. Isso aconteceu devido ao fato de termos 21 *tasks* não associadas a nenhuma *feature*, impossibilitando desta forma a realização do objetivo raiz.

Capítulo 4

Estudo de Caso

Neste capítulo apresentamos o sistema utilizado como estudo de caso deste trabalho, apresentando seu contexto e seu escopo e os resultados obtidos na aplicação da metodologia de análise de dívida técnica proposta.

4.1 Descrição do Estudo de Caso

O estudo de caso deste trabalho foi realizado no Sistema de Boletins do Exército Brasileiro (SISBOL-EB): fruto da necessidade de geração de Boletins Oficiais e Alterações dos militares. Antes do SISBOL, tal boletim era feito de maneira manual por funcionários conhecidos como *"boleteiros"*. Devido à grande quantidade de trabalho envolvida nesta atividade, alguns militares buscaram automatizar algumas etapas da geração e manutenção de boletins e alterações militares. Com o passar do tempo, novas funcionalidades foram incluídas até o momento em que o chamado "SISBOL" se tornou conhecido e foi distribuído por todo o território nacional.

O Sistema de Boletins desenvolvido durante este projeto tem como objetivo substituir sua versão legada, que é utilizada, atualmente, em todo o país. Desse modo, o escopo do novo SISBOL, desenvolvido entre o período de Outubro de 2016 a Abril de 2018, é baseado no escopo da versão legada, acrescentando algumas melhorias importantes que serão destacadas no decorrer deste tópico.

Desse modo, dada a importância deste sistema para o funcionamento de cada Organização Militar em todo país, foi observada a necessidade de padronizar e centralizar as informações de todo o Exército Brasileiro em apenas um SISBOL.

A Tabela A.1 apresenta o conjunto de features do SISBOL e a Figura 4.1 apresenta a árvore de objetivos para implementação das features apresentadas.

Em particular, as novas funcionalidades relacionadas a Workflows de Tramitação foram necessárias devido a característica de crescimento do SISBOL legado. O crescimento desestruturado do SISBOL fez com que cada Organização Militar o utilizasse de maneira distinta. Por exemplo, algumas Organizações utilizam o sistema para realizar aprovações em alguns níveis, criando um fluxo de aprovações até a publicação do Boletim. Outras Organizações utilizam o

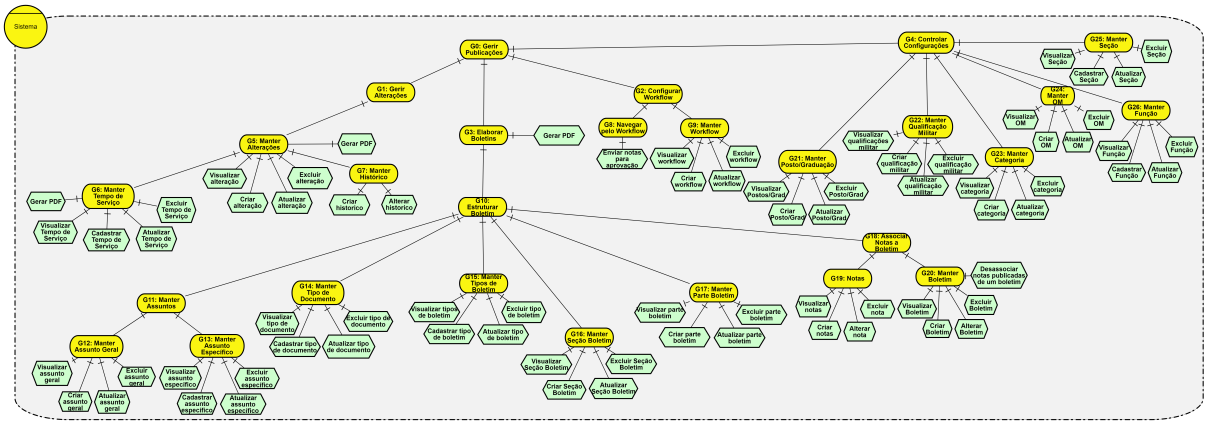


Figura 4.1: Árvore de Objetivos SISBOL

sistema apenas para registro e geração dos Boletins, inutilizando as funcionalidades de aprovação do Boletim. Desse modo, buscando agradar o máximo de Organizações Militares, foram implementadas as funcionalidades de configuração do fluxo de tramitação do Boletim até que o mesmo seja publicado. Definindo um Workflow de Tramitação, o usuário pode configurar como deseja executar o sistema, tornando-o flexível e agradável para todos os tipos de uso pelo país.

Com o objetivo de agilizar o andamento do Boletim no fluxo de tramitação incluiu-se a funcionalidade de Apresentar Pendências, a qual envolve, basicamente, a implementação de uma "caixa de entrada" com as ações pendentes para o Militar logado no sistema, além da funcionalidade de Acompanhar Citações, onde o militar é capaz de visualizar trechos onde o mesmo é citado, para acompanhar seu processo junto ao Exército.

Buscando facilitar o processo de auditoria, o qual envolvia diversos problemas na versão legada, foram incluídas as funcionalidades de Manter Histórico de Boletim e Nota, além de passar a impossibilitar a exclusão definitiva de entidades importantes do sistema, tornando-a, no máximo, inativa. Incluiu-se também a possibilidade de Pesquisar Informações nos Boletins, o que anteriormente, precisava ser feito manualmente.

A implantação deste novo SISBOL se dará de maneira centralizada, executando em apenas um local e disponibilizando as mesmas funcionalidades para todas as Organizações Militares. Além disso, o novo SISBOL inclui a ideia de contextualização, fazendo com que cada Organização Militar enxergue o SISBOL como próprio, impossibilitando que uma Organização Militar acesse Informações privadas de outra Organização. Assim, todas as informações de Boletins do Exército Brasileiro estarão centralizadas em uma única base de dados, facilitando o gerenciamento e até uma análise futura destes dados, por parte do Exército Brasileiro.

4.1.1 Características Técnicas

O novo SISBOL foi implementado utilizando uma arquitetura baseada em REST (*Representational State Transfer*), utilizando a linguagem de programação Java 1.8 para implementação do *backend* e o *framework* de *javaScript* Angular 2 para implementação do *frontend* do sistema.

O *backend* do sistema é organizado em quatro camadas com responsabilidades únicas. A camada de **Modelo** é responsável pela definição das entidades do sistema, assim como suas restrições de negócio. A camada de **Resource** é responsável por criar os *end-points* do sistema, disponibilizando os recursos de cada entidade. Já a camada de **Serviço** implementa todos os serviços de determinada entidade. Estes serviços são disponibilizados para a camada Resource, que cria o acesso a eles via REST. E a camada de **Repositório** é a responsável por gerenciar o acesso a base de dados.

Buscando garantir a qualidade do sistema, utilizou-se o conceito de BDD (*Behavior Driven Development*) para implementação dos testes e documentação do sistema, o que possibilitou a realização de pesquisas importantes relacionadas, principalmente, à rastreabilidade requisito/-código.

4.2 Coleta de Dados

Como instrumento para coleta de dados utilizou-se a técnica de artefato físico, ou seja, obtendo os com extensão *.feature* gerados na especificação de requisitos e os com extensão *.json* gerados à partir da ferramenta *Serenity* utilizada para aferir os critérios de aceitação. Os dados obtidos por estes arquivos serviram de insumo para a proposição de cálculo de dívida técnica a partir da abordagem proposta neste trabalho.

A *features* do SISBOL apresentadas na Tabela A.1 foram utilizadas como *input* para a ferramenta gerada neste trabalho. A partir da execução da ferramenta **bdd2Goal** foi possível realizar o mapeamento das *features* BDD com o modelo GORE, fato que facilitou a visualização e acompanhamento da evolução da Dívida Técnica do projeto. A atual versão da ferramenta bdd2Goal tem integração com a ferramenta chamada *Serenity*[1], para a geração dos relatórios de execução das *features* BDD.

Serenity é uma ferramenta voltada para produzir uma Documentação Viva (*Living Documentation*) da especificação e a correspondente adequação do sistema a ela (através dos resultados de execução dos respectivos testes). Possui integração com ferramentas como o *Cucumber*[2], permitindo documentar os resultados obtidos a partir da execução via *Cucumber*.

Nesta proposta, após a devida especificação das *features*, foram gerados os relatórios com a aferição do resultado da execução de cada cenário com seus respectivos passos presente na *feature*. Vale ressaltar que cada *feature* do projeto deve estar representada em um arquivo próprio de extensão *.feature*, possibilitando a fácil identificação e manipulação, por parte das ferramentas de apoio da abordagem BDD.

Além disso, a ferramenta *Serenity* disponibiliza arquivos com o resultado da execução dos testes nos formatos *.xls*, *.json* e *.html*. A Figura 4.2 apresenta um exemplo de um arquivo no formato *.json* gerado pela ferramenta *Serenity*.

[1]https://www.thucydides.info/#/
[2]https://cucumber.io/

```
{"name" :"Atualizar Tipo de Boletim Sucesso",
 "Como" :"Administrador",
 "Quero":"Atualizar Tipo de Boletim",
 "Para" :"Gerir Boletins"

   "testSteps":
   [{  "number":1,"description":"Dado que eu possuo... ",
       "duration":83, "startTime":1495484008608,
       "result":"SUCCESS"},

       {"number":2,
           "description":"E eu queira atualizar...",
           "duration":1, "startTime":1495484008691,
           "result":"SUCCESS"},

       {"number":3,
           "description":"Quando eu requisitar...",
           "duration":19, "startTime":1495484008692,
           "result":"SUCCESS"},

       {"number":4,
           "description":"Entao devo...",
           "duration":1, "startTime":1495484008711,
           "result":"SUCCESS"}],

       "userStory":{
           "id":"atualizar-tipo-de-boletim",
           "storyName":"Atualizar Tipo de Boletim",
           "path":"tipoBoletim/atualizacao/atualizar_tipo_boletim.feature",
           "type":"feature"},
```


Figura 4.2: Exemplo de arquivo exportado pelo serenity.

O relatório contêm informações da *feature*, *scenarios* e *steps*. As informações relevantes para nossa proposta são obtidas por meio da realização de um *parser* do atributo ***name*** da *feature*. Além deste, outro termo importante é o ***result*** em que seu valor (*success*, *fail* ou *skipped*) determina qual foi o resultado de execução da *feature*. Essas informações são lidas através da ferramenta **bdd2goal**, desenvolvida neste trabalho como uma evolução da ferramenta *pistar*.

4.3 Aplicação no projeto SISBOL

O processo de desenvolvimento do projeto SISBOL se baseou na utilização de *features* BDD para registro dos requisitos testáveis. Ao longo do processo de desenvolvimento foram realizadas três grandes entregas oficiais. Em relação a estas entregas, a Tabela 4.1 apresenta os dados obtidos nos relatórios datados de 13/05/2017, 01/08/2017 e 29/11/2017.

Vale ressaltar que, uma vez que os relatórios parciais gerados pela ferramenta *Serenity* são apenas os dados dos relatórios das datas supracitadas, a análise se limitou a esses dados disponíveis. Caso houvessem mais relatórios disponíveis, eles poderiam apresentar uma evolução mais bem distribuída da $satisfacao$ ao longo das entregas, como ilustrado no 3.4.2. No entanto, sendo esses dados reais do projeto, essa seção foca na análise da $satisfacao$ de requisitos conforme a abordagem BDD-GORE proposta para fins do estudo de caso.

O primeiro relatório que reporta o período de maio (13/05) de 2017, registrado um conjunto de 60 *features*, contendo um total de 171 cenários, sendo 100 com resultado positivo e 71 como *skipped*. Vale ressaltar que na primeira entrega o conjunto total de *features* foi considerado,

Tabela 4.1: Dados das entregas do SISBOL.

Relatórios	Feat.	Cen. Skip.	Cen. Fail.	Cen. Suc.	Cen. Total	Est. Atual	Est. Ideal	Satisf.	1-Satisf
13/05/2017	60	71	0	100	171	200	342	58.48%	41.52%
01/08/2017	54	61	0	86	147	172	294	58.50%	41.50%
29/11/2017	60	4	0	135	139	270	278	97.12%	2.88%

levando em consideração o projeto como um todo. Já a partir do relatório 2, foram consideradas apenas as *features* entregues na *release* em questão. Observa-se que, no primeiro relatório a "situação ideal" deveria ter 342 pontos (valor resultante das fórmulas apresentadas na Seção 3.1.2), entretanto apenas 200 pontos foram alcançados, dado que 71 cenários não obtiveram resultado positivo. Sendo assim, foi obtido um total de 58.48% de satisfação dos objetivos definidos para esta entrega, com um total de 41.52% de insatisfacao Global.

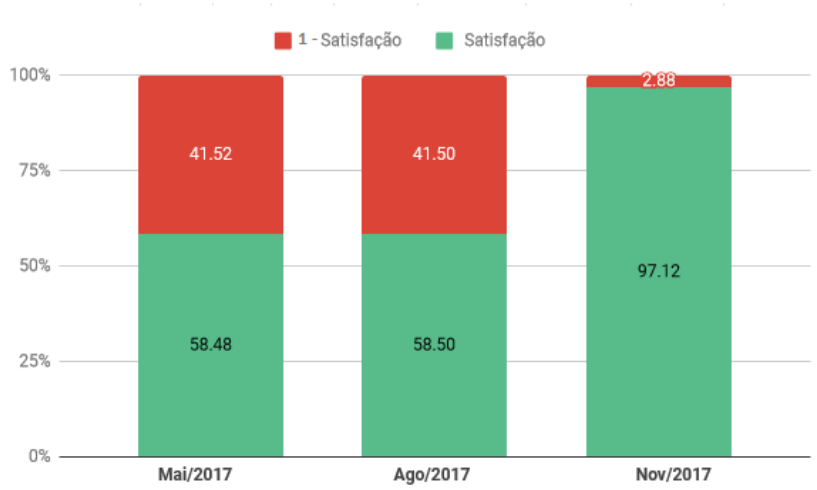


Figura 4.3: Gráfico relativo à evolução da satisfação de requisitos do projeto SISBOL nos meses mencionados.

O primeiro relatório refere-se à implementação das *features* mais simples do projeto e, por esse motivo, foi entregue um número alto de *features*, em comparação com os relatórios subsequentes. Observa-se, no entanto, que no primeiro relatório, não haviam sido levantadas os principais requisitos da nova versão do SISBOL. No segundo relatório foram implementadas 54 *features*, agregando um total de 147 cenários, sendo 86 de sucesso e 61 como *skipped*. A "situação ideal" seria um total de 294 pontos, entretanto apenas 172 foram alcançados, gerando um total de 58.50% de satisfação e 41.50% insatisfacao Global.

Já o terceiro relatório reporta o processo completo de desenvolvimento (incluindo os principais novos requisitos esperados para o SISBOL), compondo um total de 60 *features* agregando um total de 139 cenários, com apenas 4 *skipped* e 135 cenários de sucesso. A "situação ideal" neste momento seria um total de 278 pontos e 270 foram alcançados com os resultados reportados. Dessa forma, foi gerada uma satisfação de 97.12% e apenas 2.88% de insatisfação.

A Figura 4.4 apresenta a evolução da satisfação global ao longo do tempo. Apesar de ser apenas três relatórios, é possível observar a tendência de evolução da $satisfacao$ a partir da utilização da abordagem proposta neste trabalho.

4.4 Análise dos Resultados

Com base nos resultados apresentados nas seções anteriores pode-se observar que a abordagem proposta neste trabalho (BDD-GORE) possibilita o acompanhamento da satisfação de requisitos ao longo do processo de desenvolvimento de software. Os dados obtidos com a aplicação da abordagem utilizando os dados dos três relatórios do projeto SISBOL confirmam essa capacidade, apesar de possuir um número pequeno de entregas, o que dificulta a visualização mais distribuída ao longo das *sprints*. Dessa forma, fazemos a análise com base na simulação dos dados e uma análise com base nos dados reais do projeto.

4.4.1 Análise dos Dados Simulados no SISBOL

Buscando facilitar a visualização e confirmar a capacidade da abordagem de identificar e acompanhar a satisfação de requisitos ao longo do processo de desenvolvimento, foram gerados dados hipotéticos com base no projeto SISBOL, como pôde ser observado na Seção 3.1.2.

Neste sentido, o gráfico apresentado na Figura 3.4 apresenta as curvas de satisfação e de insatisfação ao longo do processo de desenvolvimento. Nesta Figura são utilizadas três configurações de processos de desenvolvimento, um contendo um caso de *sucesso*, outro representando um processo de desenvolvimento com dificuldades ao início e recuperação ao fim, e outra configuração representando um processo que não obteve sucesso ao final do desenvolvimento. Realizando uma comparação entre os dados obtidos em cada uma das configurações observa-se que a disposição das curvas de satisfação e insatisfação servem como base para acompanhamento do processo de desenvolvimento, possibilitando a compreensão da situação atual do projeto.

Complementarmente, o esperado durante um processo de desenvolvimento é que a intercessão entre as curvas de satisfação e insatisfação ocorra na metade do processo de desenvolvimento, o que pode indicar que o processo de desenvolvimento está ocorrendo de tal forma que poderá levar ao sucesso do projeto. Dessa forma, basta analisar a tendência de cada uma das curvas analisadas (satisfação e insatisfação) para compreender se o processo de desenvolvimento está seguindo o ritmo adequado ou não, possibilitando que a equipe realize adequações para retomar o ritmo correto de desenvolvimento.

Além disso, observou-se, a partir da representação no Modelo de Objetivos, listados nos Apêndices C e D, que a visualização da situação atual do projeto se torna uma atividade bastante prática e direta. Dessa forma, todos os *stakeholders* do projeto podem acompanhar com facilidade a evolução da satisfação dos requisitos BDD e a satisfação dos objetivos definidos no Modelo de Objetivos.

4.4.2 Análise dos Dados Reais do SISBOL

Para a análise dos dados reais do SISBOL, faremos uma analogia com os dados publicados do projeto [22], apresentados na Figura 4.4.2. Essa Figura apresenta o fluxo cumulativo de *commits* de desenvolvimento de código fonte e de *features* BDD por parte da equipe do SISBOL ao longo dos 16 meses do projeto. Observa-se que durante os seis primeiros meses (de 10/2016 a 04/2017) a quantidade de *commits* associados as especificações BDD no SISBOL é bastante modesta. No entanto, com a percepção dos benefícios da utilização dos testes baseados nessa abordagem, estas atividades foram incluídas no processo diário de desenvolvimento, conforme Fazzolino et al. [22].

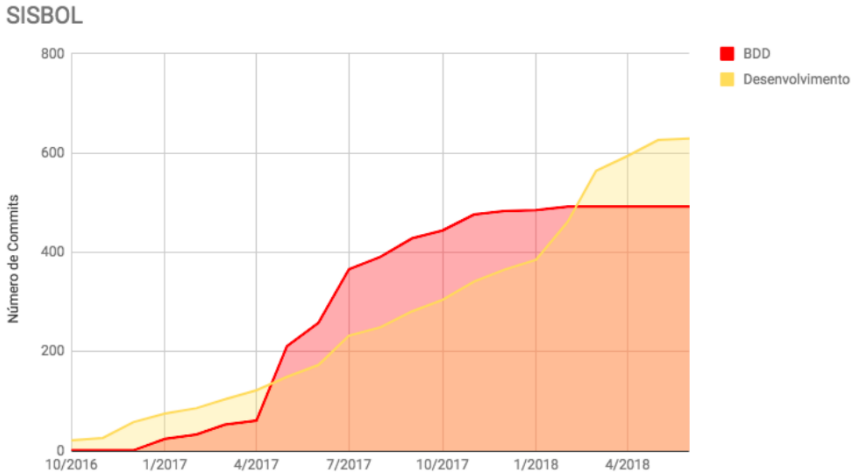


Figura 4.4: Fluxo cumulativo de commits da equipe SISBOL conforme Fazzolino et al. [22].

Percebe-se que os relatórios reportados na Tabela 4.1 estão intimamente ligados aos picos da curva cumulativa de *commits* de *features* da Figura 4.4.2: um logo após o mês 4, um outro logo após o mês 7, e por fim um logo após o mês 10. Pelos dados da nossa abordagem, a insatisfação de requisitos nesses períodos foram, respectivamente, de: $\approx 41.52\%$, $\approx 41.50\%$ e $\approx 2.88\%$. Apesar da insatisfação de requisitos ter atingido em torno de $\approx 41.52\%$ no mês de Maio, percebe-se que uma perspectiva mais fidedigna foi percebida no mês de Agosto, dada a inclinação íngreme de *commits* de *features*. Já aproximando o período do mês de Novembro (período do terceiro relatório) a curva começa a suavizar a inclinação, levando ao atingimento de $\approx 97.12\%$ da insatisfação em Novembro, conforme nossa abordagem. Percebe-se, portanto, um período de intenso desenvolvimento concentrado nos meses de Agosto a Novembro de 2017 para atingir insatisfação de requisitos. Dessa forma, o projeto SISBOL poderia refletir um pequeno atraso para o atingimento da insatisfação de requisitos em torno do meio do projeto (mês de Junho). No entanto, em uma análise mais detalhada, percebe-se que o ápice da inclinação curva cumulativa de *commits* de *features* BDD ocorre justamente nos meses de Maio e Julho de 2017.

Vale ressaltar que nos meses finais de Novembro de 2017 a Abril de 2018, não houve adição de novas *features*, apesar de um crescimento íngreme nos *commits* de desenvolvimento. Em

uma primeira análise, isso poderia indicar o esforço no grau de cumprimento da satisfação de requisitos. No entanto, segundo a equipe do projeto, apesar dos requisitos em si serem os mesmos, houve um esforço para refatoração da arquitetura, uma vez que havia necessidade de uma arquitetura mais descentralizada para fins de implantação e manutenção do SISBOL. Dessa forma, existe algum risco de os demais $\approx 2.88\%$ de insatisfação de requisitos restante não ter sido cumpridos.

4.5 Ameaças à Validade

Avaliamos a seguir as possíveis ameaças à validade do estudo caso conforme as perspectivas validade de construção, validade interna, validade externa e confiabilidade.

- *Validade de Construção*: Para a medição da $insatisfacao$ dos requisitos necessariamente devem ser especificados em *features*. A especificação em outro formato inviabilizaria a proposta, tendo em vista que se perderia a rastreabilidade entre a especificação, codificação e teste dos cenários. Outra ameaça é que para a geração dos relatórios com a aferição dos cenários foi utilizada a ferramenta *"Serenity"*, tendo com saída o arquivo no formato *".json"*. Para outras ferramentas, o formato dos arquivos e a estrutura com o resultado da execução dos cenários pode ser diferente, o que inviabilizaria a leitura pela ferramenta pistar.

  Além dessa ameaça, um outro ponto que deve ser considerado é como o modelo de objetivos do SISBOL foi construído e validado. Para mitigar essa ameaça foi realizada a validação com um dos membros da equipe de desenvolvimento responsável pela elicitação de requisitos e pela implementação das *features* BDD.

- *Validade Interna*: Utilizando nossa abordagem para o acompanhamento dos requisitos no Sistema de Boletins e Alterações foi possível realizar o mapeamento da *feature* BDD para as *tasks* da árvore de objetivos. Executamos o cálculo para cada *task* avaliando sua $insatisfacao$ local e a $insatisfacao$ global para cada *sprint*. Ao longo das entregas das *sprints* percebemos a relação entres as *features* BDD com as *tasks* do modelo de objetivos. Dessa forma ficou evidenciado como a evolução da $insatisfacao$ de requisitos foi percebida.

- *Validade Externa*: O estudo de caso realizado neste trabalho de fato foi em um sistema específico. Para fins de generalização da abordagem, torna-se necessária a realização de mais estudos para avaliação a viabilidade de generalização da abordagem proposta. No entanto, sabe-se que as características do SISBOL englobam sistemas comercialmente desenvolvidos uma vez que é um projeto real e cujos dados extraídos refletem as 16 *sprints* ao longo dos 24 meses de projeto.

- *Confiabilidade*: Uma parte considerável dos dados do estudo de caso estão apresentados no Apêndice D. Além disso, para reprodução dos resultados deste estudo de caso, dispo-

nibilizamos os dados publicamente no seguinte repositório GitHub [3] para uso e replicação do estudo de caso.

[3]https://github.com/lealfb/bdd2Goal/

Capítulo 5

Trabalhos Relacionados

Neste capítulo é feito o levantamento dos trabalhos mais relevantes que abordam (1) a utilização da abordagem BDD como forma de guiar o processo de desenvolvimento de software, bem como o gerenciamento das entregas, e (2) a utilização da abordagem GORE, também como forma de identificar o quanto os requisitos de negócio são atingidos. Após o levantamento dos principais trabalhos relacionados a esta pesquisa, destacamos as contribuições do trabalho em relação a situação atual apresentada nos trabalhos levantados.

A utilização da abordagem GORE é debatida e analisada por diversos trabalhos, tais como [20, 30, 54, 59]. Observa-se que a abordagem GORE busca aprimorar a compreensão dos requisitos envolvidos em um projeto de software. Além disso, sabe-se que um modelo de objetivos pode ser utilizado para acompanhamento e satisfabilidade dos objetivos de negócio, analisando a completude da árvore de objetivos ao longo do tempo, por exemplo.

Neste sentido, em relação ao acompanhamento dos requisitos elencados e como estes atingem os objetivos de negócio com base na abordagem GORE, destaca-se o trabalho de Ernst [20]. No trabalho de Ernst é utilizado o termo "Dívida Técnica" para acompanhar a evolução dos requisitos ao longo do processo de desenvolvimento. Ele propõe uma ferramenta para identificação e gerenciamento da dívida técnica de requisitos, denominada *RE-KOMBINE*. Tal ferramenta se baseia na utilização de modelos de objetivos para identificação e acompanhamento da evolução do projeto, tendo como foco a identificação da dívida técnica de requisitos. A partir do modelo de objetivos a ferramenta busca responder as seguintes questões: (1) dado um conjunto de *tasks* S, este conjunto satisfaz os requisitos G? (2) Dado um conjunto de requisitos G, qual conjunto de *tasks* S que satisfaz estes requisitos? e (3) Dada uma mudança de requisitos, quais novas *tasks* são necessárias para satisfazer a alteração nos requisitos?

Em comparação com a pesquisa apresentada neste trabalho, observa-se que tal abordagem baseia-se na utilização do modelo de objetivos para visualização e acompanhamento das *tasks* que levam à conclusão do produto de software, ou seja, alcançam as metas especificadas no modelo de objetivos. Entretanto, a ferramenta *RE-KOMBINE*, proposta por [20], carece de estratégias que facilitem a automatização do rastreio dos requisitos no código fonte do software. Esta carência é levantada pelo próprio autor como uma lacuna que deverá ser sanada em trabalhos futuros.

Neste sentido, a abordagem proposta neste trabalho apresenta uma estratégia para cobrir tal lacuna: a utilização da abordagem BDD em conjunto com o modelo de objetivos, possibilitando a automação dos requisitos especificados utilizando *features* BDD, a relação da *feature* com o código fonte que a implementa e o mapeamento da *feature* com a *tasks* do modelo de objetivos.

Vale destacar, também, a utilização do modelo GORE em conjunto com o conceito de história de usuário, apresentado em [57] e [58]. Em ambos trabalhos, Wautelet et al. propuseram a geração automatizada de um modelo de objetivos baseada em histórias de usuário. Dessa forma, a partir da definição de uma história de usuário, os *stakeholders* do projeto são capazes de visualizar, compreender e debater os requisitos a partir de uma representação gráfica estruturada na forma de árvore de objetivos. Para tal, Wautelet et al. leva em consideração um *template* bem definido de história de usuário, de modo que a história possa ser processada de maneira automatizada, extraindo algumas informações a partir das representações de "*Who, What* e *Why*", conceitos comuns no padrão de histórias de usuário. Dessa forma, passa a ser possível visualizar uma história de usuário de tal forma que facilita a identificação de tarefas e objetivos com maior granularidade e clareza.

Esta discussão sobre relacionar artefatos do modelo de objetivos com artefatos de requisitos ágeis também foi debatida no trabalho de [46], onde foi apresentada uma linguagem chamada de *Scrum i**, buscando facilitar o entendimento do contexto do software em desenvolvimento e buscando suprir a carência no *Scrum* em relação as relações de dependência entre atores.

Seguindo esta linha, a proposta de Wautelet et al. [58] traz benefícios importantes para o processo ágil de desenvolvimento de software, possibilitando a visualização, discussão e disseminação do conhecimento sobre os requisitos com todos os *stakeholders* do projeto. Isto ocorre tanto a partir da utilização de histórias de usuário comuns, quanto a partir da utilização de representações GORE das histórias de usuários. Entretanto, é importante destacar que a proposta de Wautelet et al. [58] só leva em consideração os artefatos de requisitos, ou seja, as histórias de usuário e o modelo GORE gerado. Dessa forma, a rastreabilidade do requisito com sua implementação não é suportado.

Assim sendo, observando a lacuna deixada pelo trabalho discutido acima, a abordagem BDD-GORE busca possibilitar, além do mapeamento do requisito (neste trabalho descrito no formato BDD) e um modelo de objetivos (GORE), busca também possibilitar o acompanhamento do processo de desenvolvimento de cada um dos reqquisitos e *tasks* relacionadas. Dessa forma, pode-se visualizar o conjunto de requisitos no modelo de objetivos e visualizar a evolução do *status* do projeto, ou seja, acompanhar a implementação de cada requisito de maneira automatizada e contínua, mantendo o conceito de "*Living Documentation*".

Em relação a utilização da abordagem BDD como forma de guiar o processo de desenvolvimento, destacam-se os trabalhos de [32, 45, 50, 52]. Os benefícios da adoção da abordagem BDD fazem parte de um consenso por parte dos pesquisadores da área. Entre os benefícios citados comumente, destacam-se: (1) melhoria na comunicação entre os *stakeholders* do projeto, (2) acesso prático e direto à rastreabilidade dos artefatos do projeto de software (requisitos, código e testes) e (3) *Living Documentation*.

Dessa forma, seguindo a lógica apresentada por Wautelet et al. [58], esta pesquisa busca unir as vantagens do mapeamento dos requisitos com a adoção da abordagem BDD e o modelo de objetivos. Dessa forma, os requisitos são definidas seguindo o padrão BDD, mapeados para um modelo de objetivos e acompanhadas a cada entrega a partir do processamento das *features* BDD. Ou seja, utilizando a abordagem BDD-GORE, pode-se acompanhar a evolução do processo de desenvolvimento a cada *sprint* a partir de uma representação GORE dos artefatos BDD, levando em consideração o *status* atual de cada cenário BDD (*Success, Fail, Skipped*).

Com o objetivo de destacar diferenças e semelhanças entre os trabalhos destacados neste Capítulo, apresentamos a Tabela 5.1.

Tabela 5.1: Comparação entre trabalhos relacionados.

Trabalho	**Acompanhamento do Processo de desenvolvimento**	**Mapeamento dos requisitos no GORE**	**Atualização do modelo GORE**
Ernst [20]	Sim	Manual	Manual
Wautelet et. al. [58]	Sim	Automatizado	Manual
BDD-GORE	Sim	Automatizado	Automatizado

Capítulo 6

Conclusão

O processo de desenvolvimento de software vem sendo evoluído e aprimorado continuamente, seja a partir da definição de novas abordagens, técnicas e ferramentas, ou a partir da junção de abordagens, ferramentas e técnicas. Dessa forma, este trabalho se baseou no conceito de *Living Documentation* para buscar a aprimorar a gestão do processo de desenvolvimento. Para isso, foram utilizados os conceitos de BDD e GORE, servindo como matéria-prima para definição de uma abordagem que viabilize a medição e o acompanhamento do projeto ao longo do processo de desenvolvimento.

De acordo com o apresentado ao longo do trabalho, alguns desafios do processo de desenvolvimento são atacados com a definição da abordagem proposta neste trabalho. Entre estes desafios, destacam-se a comunicação entre os *stakeholders*, elicitação dos requisitos , a rastreabilidade entre os requisitos e o código-fonte e, ainda, o acompanhamento de projetos ágeis ao longo do processo de desenvolvimento de software.

A partir do desenvolvimento deste trabalho foi possível conceber uma metodologia para apoiar a gestão do projeto de auxiliando a engenharia de requisitos em projetos ágeis, apresentando um mapeamento de um conjunto *features* BDD para um modelo modelo de objetivos i*. Este mapeamento do modelo de *features* BDD para a abordagem GORE nos forneceu uma forma de identificar e acompanhar a evolução dos requisitos do projeto e a definição de uma função para seu cálculo. Buscando analisar a viabilidade da utilização da ferramenta e aplicação da técnica de acompanhamento do projeto proposta, utilizou-se, neste trabalho, o projeto SISBOL apresentado no Capítulo 4.1.

Os resultados apresentados ao longo do trabalho evidenciam a contribuição desta pesquisa, dado que a estratégia de cálculo e acompanhamento do projeto, assim como as abordagens utilizadas para tal, agregam valor ao processo de desenvolvimento com a inclusão de um esforço mínimo (apenas algumas horas para criação do Modelo de Objetivos e relacionamento com as *features* BDD) para a equipe de desenvolvimento. Dessa forma, observa-se que a abordagem apresentada neste trabalho pode colaborar significativamente com projetos de desenvolvimento de software, buscando maximizar a chance de sucesso do projeto.

6.1 Contribuições

A contribuição mais importante deste trabalho é a elaboração de uma abordagem para para apoiar a gestão do projeto através de modelos visuais. Também podemos apresentar as seguintes contribuições desta proposta:

- forma de documentação visual em projetos ágeis;
- melhoria no entendimento dos objetivos de negócio do sistema a ser desenvolvido;
- melhoria da rastreabilidade das *features* BDD a partir de um modelo visual;
- melhoria do processo tomada de decisão de acordo com os requisitos, levando em conta o entendimento dos objetivos negociais e ao raciocínio do por quê de um requisito uma vez que estão descritos no modelo de objetivos;
- suporte ferramental, possibilitando a automatização do mapeamento das *feature* BDD e modelo de objetivos.

6.2 Trabalhos Futuros

Com o objetivo de dar continuidade à pesquisa desenvolvida por esta dissertação, destacamos os seguintes trabalhos futuros:

- Exploração de outras características do modelo de objetivos.
- melhoria da visualização da evolução do projeto.
- elaboração de diretrizes para mapeamento do modelo de objetivos para as *features* BDD.
- execução de outros estudos de caso comparando a proposta identificando possíveis melhorias.
- definição de um processo de engenharia de requisitos para métodos ágeis envolvendo *features* BDD e modelos de objetivos.

Referências

[1] Abad, Zahra Shakeri Hossein e Guenther Ruhe: *Using real options to manage technical debt in requirements engineering.* Em *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, páginas 230–235. IEEE, 2015. 3

[2] Adzic, Gojko: *Specification by example: how successful teams deliver the right software.* Manning Publications Co., 2011. 1, 10, 11, 13

[3] Ågerfalk, J, Brian Fitzgerald e Old Petunias In: *Flexible and distributed software processes: old petunias in new bowls.* Em *Communications of the ACM*. Citeseer, 2006. 8

[4] Al-Zewairi, Malek, Mariam Biltawi, Wael Etaiwi e Adnan Shaout: *Agile software development methodologies: survey of surveys.* Estonian Journal of Earth Sciences, 67(3):74–98, 2018. 9

[5] Beck, Kent: *Extreme programming explained: embrace change.* addison-wesley professional, 2000. 7

[6] Beck, Kent: *Test-driven development: by example.* Addison-Wesley Professional, 2003. 2, 11

[7] Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries *et al.*: *Manifesto for agile software development.* 2001. 7

[8] Beedle, Mike, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Ken Schwaber, Jeff Sutherland e Dave Thomas: *Manifesto Ágil*. `https://agilemanifesto.org/`, acesso em 2018-12-02. 9

[9] Behutiye, Woubshet Nema, Pilar Rodríguez, Markku Oivo e Ayşe Tosun: *Analyzing the concept of technical debt in the context of agile software development: A systematic literature review.* Information and Software Technology, 82:139–158, 2017. 9

[10] Boehm, Barry: *Get ready for agile methods, with care.* Computer, 35(1):64–69, 2002. 8

[11] Cao, Lan e Balasubramaniam Ramesh: *Agile requirements engineering practices: An empirical study.* IEEE software, 25(1), 2008. 1, 9, 10

[12] Chung, Lawrence, Brian A Nixon, Eric Yu e John Mylopoulos: *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012. 15

[13] Cucumber: *Gherkin reference*. `https://docs.cucumber.io/gherkin/reference/`, acesso em 2019-09-01. 2

[14] Dardenne, Anne, Axel Van Lamsweerde e Stephen Fickas: *Goal-directed requirements acquisition*. Science of computer programming, 20(1-2):3–50, 1993. 15

[15] Dick, Jeremy, Elizabeth Hull e Ken Jackson: *Requirements engineering*. Springer, 2017. 9

[16] Dikert, Kim, Maria Paasivaara e Casper Lassenius: *Challenges and success factors for large-scale agile transformations: A systematic literature review*. Journal of Systems and Software, 2016. 8

[17] Domain Driven DesignSantos, E C S A study of test techniques for integration with, D M Beder e R A D Penteado: *A study of test techniques for integration with Domain Driven Design*. 2015 12th International Conference on Information Technology - New Generations, páginas 373–378, 2015. 13, 15

[18] Dybå, Tore e Torgeir Dingsøyr: *Empirical studies of agile software development: A systematic review*. Information and software technology, 50(9):833–859, 2008. 8

[19] Erickson, John, Kalle Lyytinen e Keng Siau: *Agile modeling, agile software development, and extreme programming: the state of research*. Journal of database Management, 16(4):88, 2005. 7

[20] Ernst, Neil A: *On the role of requirements in understanding and managing technical debt*. Em *Proceedings of the Third International Workshop on Managing Technical Debt*, páginas 61–64. IEEE Press, 2012. 1, 4, 43, 45

[21] Evans, Eric: *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004. 2, 12

[22] Fazzolino, Rafael, Henrique Medrado de Faria, Luis Henrique Vieira Amaral, Edna Dias Canedo, Genaína Nunes Rodrigues e Rodrigo Bonifácio: *Assessing agile testing practices for enterprise systems: A survey approach*. Em *SAST*, páginas 29–38. ACM, 2018. xi, 40

[23] Franch, Xavier: *The i framework: The way ahead*. Em *Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on*, páginas 1–3. IEEE, 2012. 16

[24] Inayat, Irum, Siti Salwah Salim, Sabrina Marczak, Maya Daneva e Shahaboddin Shamshirband: *A systematic literature review on agile requirements engineering practices and challenges*. Computers in human behavior, 51:915–929, 2015. 10, 11

[25] io, Cucumber: *Gherkin reference*. `https://docs.cucumber.io/gherkin/reference/`, acesso em 2018-12-02. xi, 14

[26] io, Cucumber: *Introduction to tdd and bdd*, 2017. `https://cucumber.io/blog/2017/05/15/intro-to-bdd-and-tdd`, acesso em 2018-12-01. xi, 12

[27] Kolp, Manuel, Paolo Giorgini e John Mylopoulos: *A goal-based organizational perspective on multi-agent architectures*. Em *International Workshop on Agent Theories, Architectures, and Languages*, páginas 128–140. Springer, 2001. 2

[28] Korkala, Mikko: *Customer communication in distributed agile software development*. 2015. 9

[29] Laplante, Phillip A: *Requirements engineering for software and systems*. Auerbach Publications, 2017. 1

[30] Lapouchnian, Alexei: *Goal-oriented requirements engineering: An overview of the current research*. University of Toronto, página 32, 2005. 43

[31] Lee, Gwanhoo e Weidong Xia: *Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility*. Mis Quarterly, 34(1):87–114, 2010. 9

[32] Lucassen, Garm, Fabiano Dalpiaz, Jan Martijn EM van der Werf, Sjaak Brinkkemper e Didar Zowghi: *Behavior-driven requirements traceability via automated acceptance tests*. Em *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, páginas 431–434. IEEE, 2017. 2, 44

[33] Martin, Robert C: *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002. 1

[34] Mazuco, Alan SC: *Percepções de práticas Ágeis em desenvolvimento de software: Benefícios e desafios*, 2017. xi, 8

[35] Melo, C, Viviane A Santos, Hugo Corbucci, Eduardo Katayama, Alfredo Goldman e Fabio Kon: *Métodos ágeis no brasil: estado da prática em times e organizações*. São Paulo: Departamento de Ciência da Computação do IME/USP, 2012. 8

[36] Mendes, Thiago Souto, Mário André de F Farias, Manoel Mendonça, Henrique Frota Soares, Marcos Kalinowski e Rodrigo Oliveira Spínola: *Impacts of agile requirements documentation debt on software projects: a retrospective study*. Em *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, páginas 1290–1295. ACM, 2016. 1

[37] Nerur, Sridhar, RadhaKanta Mahapatra e George Mangalaraj: *Challenges of migrating to agile methodologies*. Communications of the ACM, 48(5):72–78, 2005. 8

[38] North, Dan: *Introducing bdd*, 2003. `https://dannorth.net/introducing-bdd/`, acesso em 2018-06-15. 11, 12

[39] North, Dan: *Introducing BDD*, 2015. `https://dannorth.net/introducing-bdd/`, `https://dannorth.net/introducing-bdd/`. 2

[40] Paetsch, Frauke, Armin Eberlein e Frank Maurer: *Requirements engineering and agile software development*. Em *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, páginas 308–313. IEEE, 2003. 9

[41] Pressman, Roger S: *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005. 16

[42] Ramesh, Balasubramaniam, Lan Cao e Richard Baskerville: *Agile requirements engineering practices and challenges: an empirical study*. Information Systems Journal, 20(5):449–480, 2010. xiii, 10

[43] Santander, Victor Francisco Araya, André Abe Vicente, Fabio G Köerich e Jaelson Brelaz de Castro: *Modelagem de requisitos organizacionais, não-funcionais e funcionais em software legado com ênfase na técnica i**. Em *CIbSE*, páginas 47–60, 2007. 2

[44] Santos Carvalho, Francisco do: *Modelagem organizacional e gestão do conhecimento: o caso da universidade estadual do sudoeste da bahia*. 2003. 16

[45] Scandaroli, André, Rodrigo Leite, Aléxis H Kiosia e Sandro A Coelho: *Behavior-driven development as an approach to improve software quality and communication across remote business stakeholders, developers and qa: two case studies.* Em *Proceedings of the 14th International Conference on Global Software Engineering*, páginas 95–100. IEEE Press, 2019. 44

[46] Scheidegger, MES: *Integrando Scrum e a Modelagem de Requisitos Orientada a Objetivospor meio do SCRUM i*.* Tese de Doutoramento, Dissertação de Mestrado. UFPE–CIN, 2011. 44

[47] Schwaber, Ken: *Agile project management with Scrum.* Microsoft press, 2004. 7

[48] Silva, MJ: *U-TROPOS: uma proposta de processo unificado para apoiar o desenvolvimento de software orientado a agentes.* Tese de Doutoramento, Dissertação de Mestrado, Universidade Federal do Pernambuco, 2008, 191p, 2008. xi, 17, 18

[49] Smart, John Ferguson: *BDD in Action.* Manning Publications, 2014. 12

[50] Smart, John Ferguson: *BDD In Action Behavior-Driven Development for the whole software lifecycle.* Número December. 2014, ISBN 9781617291654. 44

[51] Soares, Henrique F, Nicolli SR Alves, Thiago S Mendes, Manoel Mendonça e Rodrigo O Spínola: *Investigating the link between user stories and documentation debt on software projects.* Em *2015 12th International Conference on Information Technology-New Generations*, páginas 385–390. IEEE, 2015. 3

[52] Solis, Carlos e Xiaofeng Wang: *A study of the characteristics of behaviour driven development.* Em *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, páginas 383–387. IEEE, 2011. 2, 11, 12, 13, 44

[53] Thangasamy, S: *Lessons learned in transforming from traditional to agile development.* 2012. 9

[54] Van Lamsweerde, Axel: *Goal-oriented requirements engineering: A guided tour.* Em *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, páginas 249–262. IEEE, 2001. `http://ieeexplore.ieee.org/abstract/document/948567/`, acesso em 2017-08-02. 2, 15, 43

[55] Van Lamsweerde, Axel: *Goal-oriented requirements engineering: A guided tour.* Em *Proceedings fifth ieee international symposium on requirements engineering*, páginas 249–262. IEEE, 2001. 4

[56] Van Lamsweerde, Axel: *Requirements engineering: From system goals to UML models to software*, volume 10. Chichester, UK: John Wiley & Sons, 2009. 15

[57] Wautelet, Yves, Samedi Heng e Manuel Kolp: *Perspectives on user story based visual transformations.* Em *REFSQ Workshops*, 2017. 44

[58] Wautelet, Yves, Samedi Heng, Manuel Kolp, Isabelle Mirbel e Stephan Poelmans: *Building a rationale diagram for evaluating user story sets.* Em *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, páginas 1–12. IEEE, 2016. 44, 45

[59] Yu, Eric: *Modelling strategic relationships for process reengineering.* Social Modeling for Requirements Engineering, 11:2011, 2011. 15, 43

[60] Yu, Eric Siu Kwong: *Modelling Strategic Relationships for Process Reengineering.* Tese de Doutoramento, Toronto, Ont., Canada, Canada, 1995, ISBN 0-612-02887-9. AAINN02887. 2, 15, 16, 17

Apêndice A

Features SISBOL

Este Apêndice apresenta o conjunto de *features* envolvidas no processo de desenvolvimento do SISBOL.

Tabela A.1: Features do SISBOL.

Features		
Criar assunto especifico	Visualizar assunto especifico	Criar notas para boletim
Criar assunto geral	Visualizar assunto geral	Criar boletins
Criar posto/graduação	Visualizar postos/graduações	Excluir nota
Criar tipo de documento	Visualizar tipos de documento	Alterar notas
Criar qualificação Militar	Visualizar qualificações militar	Gerar pdf de nota
Visualizar Militar	Excluir assunto especifico	Visualizar função
Excluir Militar	Atualizar assunto especifico	Excluir função
Atualizar Militar	Excluir assunto geral	Atualizar função
Cadastrar Militar	Atualizar assunto geral	Cadastrar função
Visualizar parte Boletim	Excluir posto/graduação	Enviar notas para aprovação
Excluir parte Boletim	Atualizar posto/graduação	Enviar Notas para publicação
Atualizar parte Boletim	Excluir tipo de documento	Excluir Nota do Boletim
Criar parte Boletim	Atualizar tipo de documento	Enviar Notas para o Boletim
Visualizar Seção	Excluir qualificação militar	Excluir boletim
Atualizar Seção	Atualizar qualificação militar	Alterar boletins
Excluir Seção	Visualizar tipos de boletim	Associar notas publicadas Boletim
Cadastrar Seção	Excluir tipo de boletim	Criar Seção Boletim
Visualizar Organização Militar	Atualizar tipo de boletim	Visualizar boletim
Excluir Organização Militar	Criar tipo de boletim	Excluir Seção Boletim
Atualizar Organização Militar	Visualizar categoria	Atualizar Seção Boletim
Criar Organização Militar	Excluir categoria	Criar categoria
Visualizar Seção boletim	Atualizar categoria	Visualizar notas

Apêndice B

Instalação bdd2Goal

Este apêndice descreve o passo a passo da instalação do bdd2Goal

B.1 Instalação

B.1.1 Passo a passo de instalação

Observação Importante: O procedimento abaixo leva em consideração que a ferramenta vai rodar em um servidor **GNU/linux**, ou seja, a sequência de comandos listados abaixo são para distribuições Linux, e usaremos os padrões da distribuição Ubuntu como exemplo.

B.1.2 Instalação do Apache HTTP Server

Vamos iniciar, com a instalação do servidor HTTP. Para instalar o Apache, execute o comando abaixo:

```
apt install apache2
```

Lembre-se que a senha de administrador do sistema é necessária para executar o comando.

B.1.3 Clonando o Repositório do Github

Agora, assumindo que você já tenha o Apache HTTP Server com serviço em execução, faça uma clonagem do repositório do projeto com o comando:

```
git clone https://github.com/lealfb/bdd2Goal.git
```

Uma vez baixado o repositório, faça uma cópia do diretório "tool"e todos seu conteúdo para a diretório de publicação do Apache. Por padrão no Ubuntu Linux, o diretório fica localizado no endereço "/var/www/html". Caso tenha outra configuração no seu servidor, altere o destino conforme necessário para o seu sistema.

```
cp -R bdd2Goal/tool /var/www/html/pistar
```

Lembre-se que você deve ter permissão para copiar o conteúdo para o diretório de destino (usuário root). Após finalizada a cópia do diretório, é necessário também definir as permissões de leitura e escrita para o mesmo e seus itens subsequentes:

```
chown -R www-data:www-data /var/www/html/pistar
```

B.1.4 Estrutura de Funcionamento

Com base no projeto original, foi desenvolvido um plugin, localizado no endereço do projeto “app/ui/pistarplugin.js”. Neste arquivo estão contidas novas funções as quais sobreescrevem o funcionamento original do piStar. o qual é responsável por carregar a lista de tarefas.

Além do arquivo de plugin, também existe um novo diretório chamado *json*, o qual contém diversos arquivos JSON que é a base para funcionamento do novo tipo de propriedade contendo as taferas. Um novo script de plugin sobre-escreve as propriedades do funcionamento original do piStar e carrega valores pré-definidos desse diretório contendo os arquivos JSON.

Com o serviço em execução, acesse o endereço do servidor em “http://localhost/pistar”, o sistema deve apresentar a página de entrada conforme a figura abaixo:

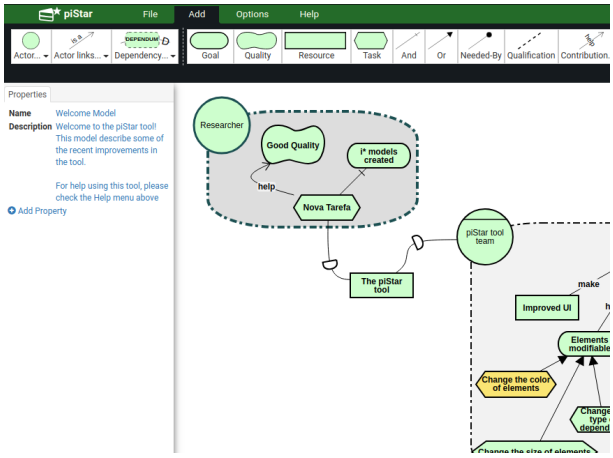


Figura B.1: Adicionando um novo atributo.

Ao selecionar uma tarefa (objeto *task*), e adicionar um novo atributo, o sistema deve mostrar ao invés de um campo livre, um campo de seleção única contendo a lista carregada a partir do diretório *json* na raiz do projeto, conforme a imagem abaixo.

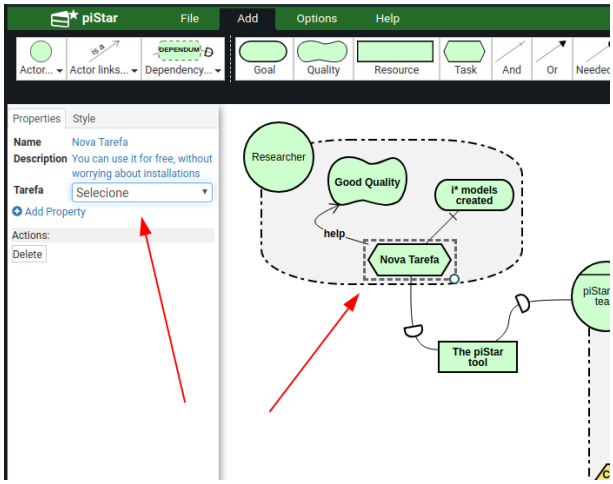


Figura B.2: Lista de tarefas.

Ao adicionar uma nova propriedade num objeto do tipo "tarefa (*task*)", deverá aparecer um menu contendo as tarefas pré-definidas do diretório contendo os diversos arquivos *json*.

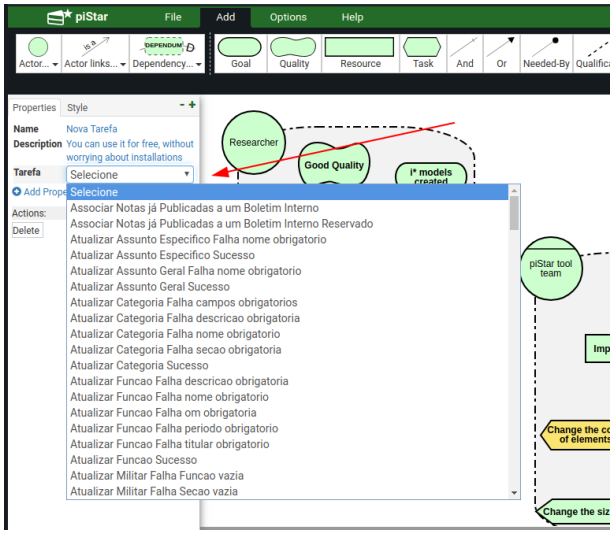


Figura B.3: Lista de tarefas expandida.

Adicionando mais um novo atributo, o sistema irá adicionar um novo combobox contendo uma lista de nível de complexidade.

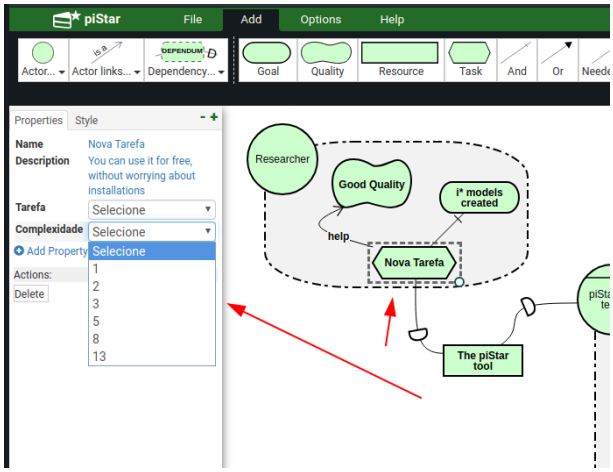


Figura B.4: Lista de complexidade.

B.1.5 Alternativa de carregamento dos arquivos *json*

Foi desenvolvido também uma alternativa de carregamento dos arquivos *json* para que o próprio usuário forneça conforme seja necessário. Para isso, um novo botão de *upload* múltiplo foi adicionado à aba "File" do sistema, conforme a figura abaixo.

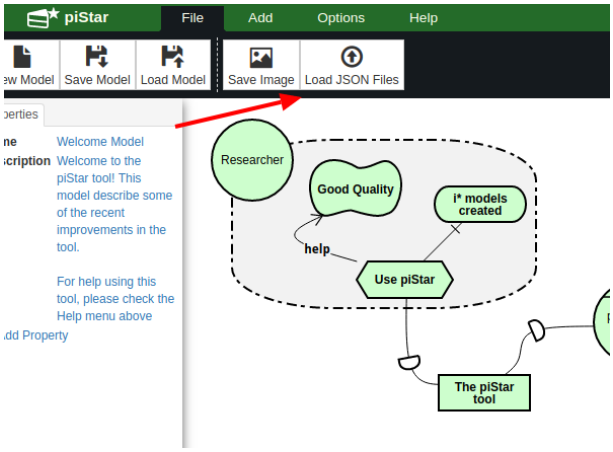


Figura B.5: Novo botão para upload dos arquivos JSON manualmente.

Ao acionar o botão, o sistema abre um *popup* com um botão para *upload* dos arquivos *json*. Estes arquivos são carregados assim que for acionado o botão enviar, zerando a base de tarefas e criando uma nova. A figura abaixo mostra o procedimento de *upload*.

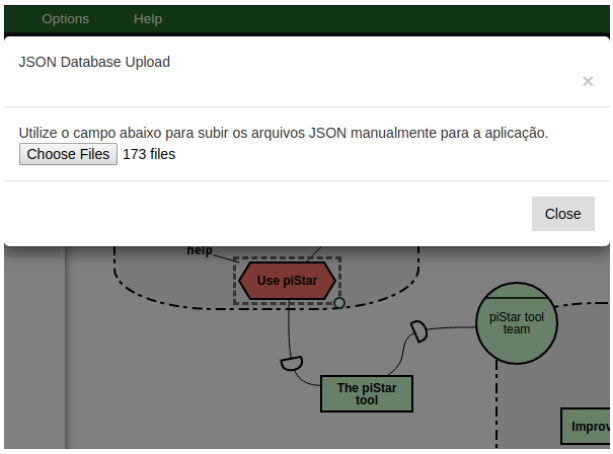


Figura B.6: Pop-up contendo formulário para upload dos arquivos.

B.1.6 Identificação do Resultado Através de Cores

Como cada *task* tem um resultado gerado armazenado ao final do arquivo *json*, foi feita também uma evolução no *plugin*, propondo que ao atribuir um valor para um objeto do

tipo *task*, automaticamente a cor do objeto muda seguindo o padrão de cores proposto a seguir:

- **SUCCESS**: Verde
- **FAILURE**: Vermelho
- **PENDING**: Azul

Logo no início do arquivo do *plugin* temos um vetor contendo objetos que definem as chaves de resultado e seus respectivos valores hexadecimais para cada situação.

```
var colours = [
    {colour: 'SUCCESS', hex: '#6CFA4B'},
    {colour: 'FAILURE', hex: '#FA7267'},
    {colour: 'SKIPPED', hex: '#B4B8FA'},
];
```

Na Figura B.7 é apresentado um caso ***Success***.

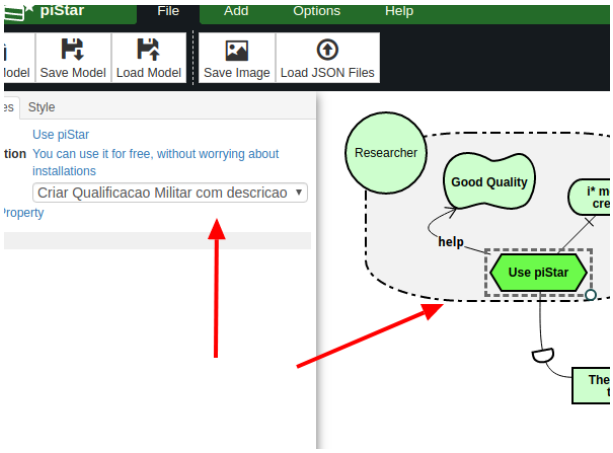


Figura B.7: Task com resultado tipo *Success*.

Na Figura B.8 é apresentado um caso tipo ***Failure***.

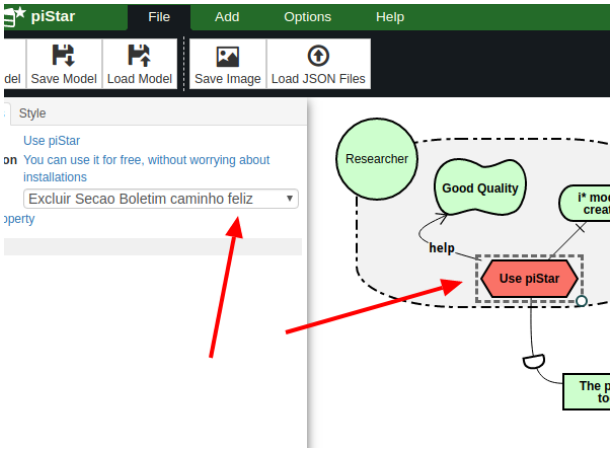


Figura B.8: Task com resultado tipo *Failure*.

Na Figura B.9 é apresentado um caso ***Skipped***

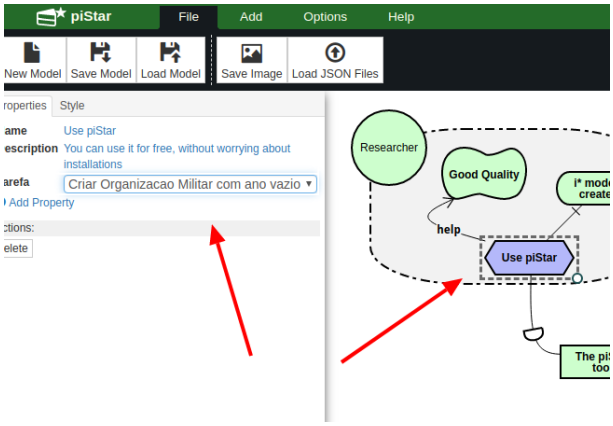


Figura B.9: Task com resultado tipo *Skipped*.

B.2 Como funciona o plugin?

Primeiramente, o plugin inicia uma requisição Ajax para listar o diretório JSON do pistar, com a função abaixo:

```
$.ajax({
    url: "json/",
    success: function (data) {
        $(data).find("a").each(function (a, b) {
            if (/.+\.json/.test(b.href)) {
                var href = b.href;
                var base = href.replace(/\/[a-f|A-F|0-9]+.json$/, '');
                var href = href.replace(base, '');
                fList.push('json' + href);
            }
        });
    }
});
function loadNames() {
    fList.forEach(function (jsonFile) {
        $.getJSON(jsonFile, function (data) {
            titulos.push(data.name);});});
}
```

Uma das necessidades de utilizar o servidor *http* em execução é para poder listar o diretório com os arquivos de forma remota, sem violar as restrições do protocolo SOAP. Após ser feita a requisição, cada arquivo *json* do diretório é aberto e lido para armazenar os atributos "name" e "result" do mesmo, o qual é a lista de tarefas pré-definida no projeto.

B.2.1 Sobrescrevendo o código original do piStar

No trecho *javascript* abaixo, começamos a sobrescrever o código original do piStar com o funcionamento necessário. Foi utilizada essa abordagem para não alterar a estrutura original do projeto e flexibilizar a possibilidade de atualização de código sem alterar o código fonte original do projeto base. Como podemos ver, é um atributo do objeto do piStar recebendo uma nova função como atributo, realizando assim a sobrescrita.

```
ui.components.PropertiesTableView.prototype.renderCustomProperty =
```

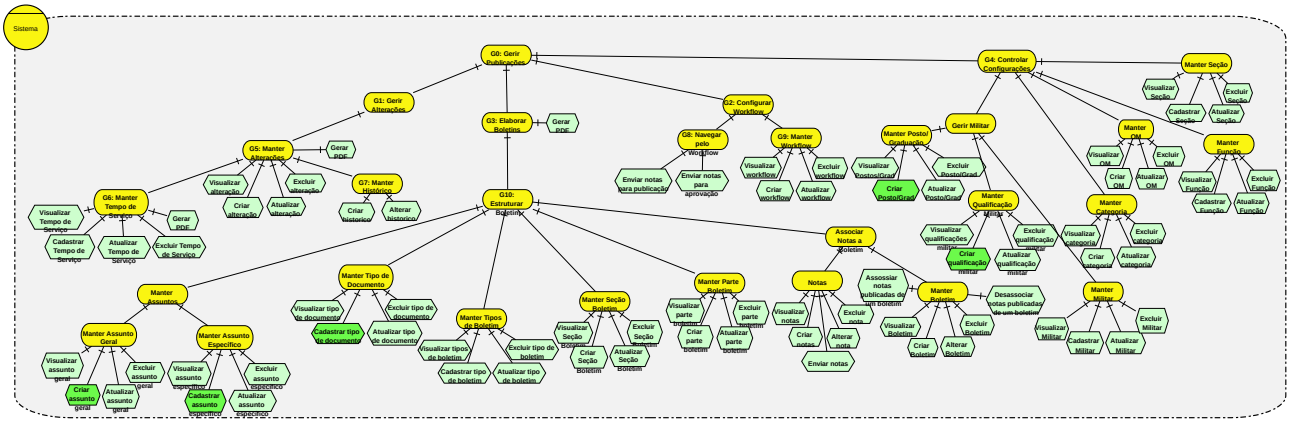
```
function (propertyName) {
    if (this.model.attributes.type == 'Task') {
        var customProperties = this.model.attributes.customProperties;
        var keys = Object.keys(customProperties);
        var customTemplate = null;
        switch (keys.indexOf(propertyName)) {
            case 1:
                customTemplate = renderCustomPropertyTemplate(propertyName,
                this.model.prop('customProperties/' + propertyName));
                break;
            case 2:
                customTemplate = renderComplexityTemplate(propertyName,
                this.model.prop('customProperties/' + propertyName));
                break;
            default:
                customTemplate = this.template({
                    propertyName: propertyName,
                    propertyValue: this.model.prop('customProperties/' + propertyName
                    dataType: 'textarea'
                });
        }
    } else {
        this.$table.find('tbody').append(this.template({
            propertyName: propertyName,
            propertyValue: this.model.prop('customProperties/' + propertyName),
            dataType: 'textarea'
        }));
    }
    this.$table.find('tbody').append(customTemplate);
};
```

Apêndice C

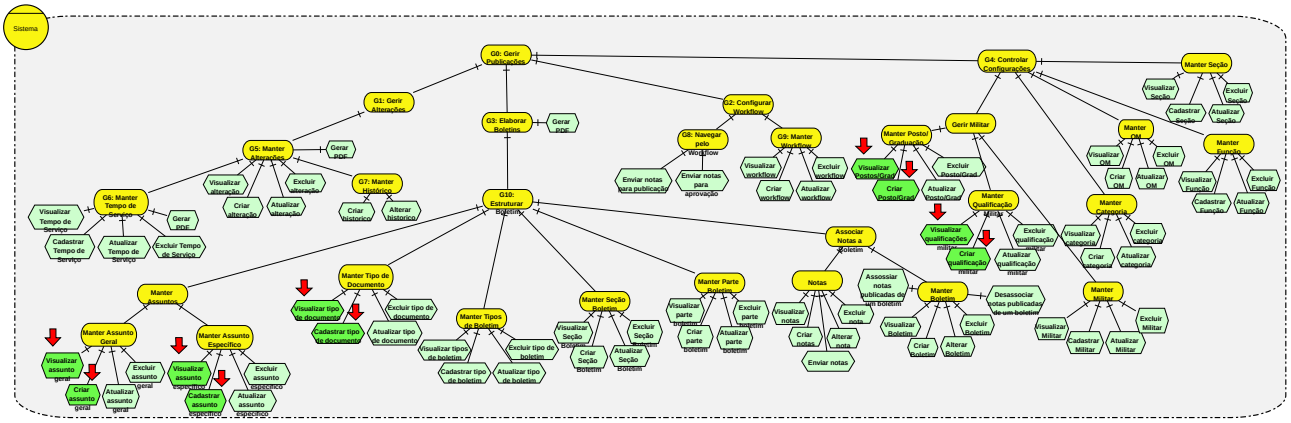
Evolução da Árvore de Objetivos

C.1 Evolução da Árvore de Objetivos

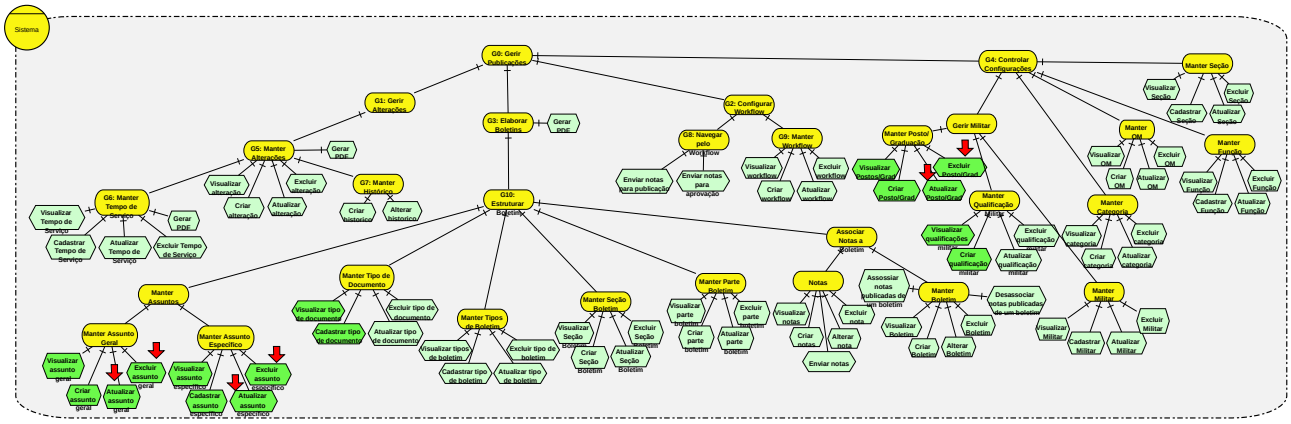
C.1.1 Árvore de Objetivos – Início do projeto



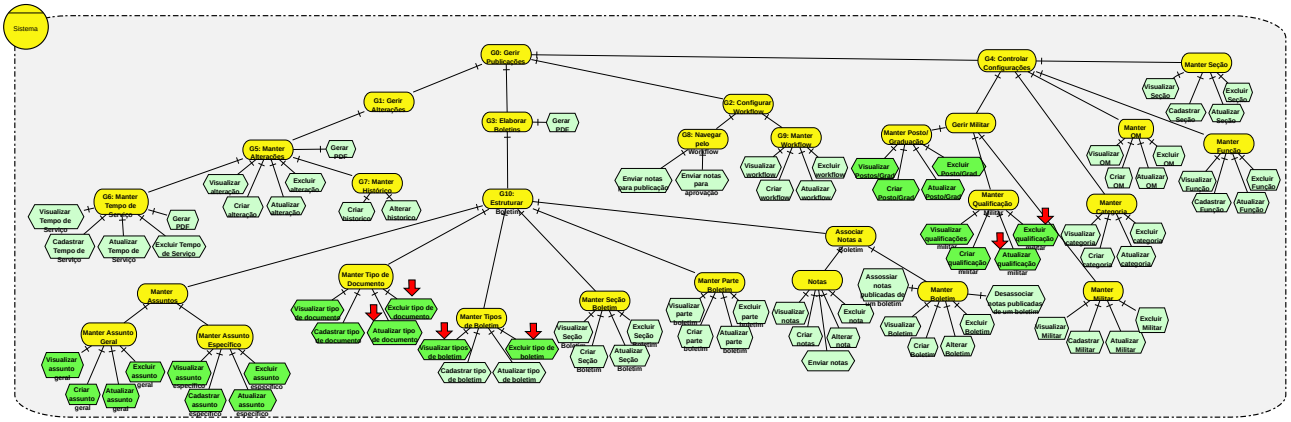
C.1.2 Árvore de Objetivos – Sprint 2



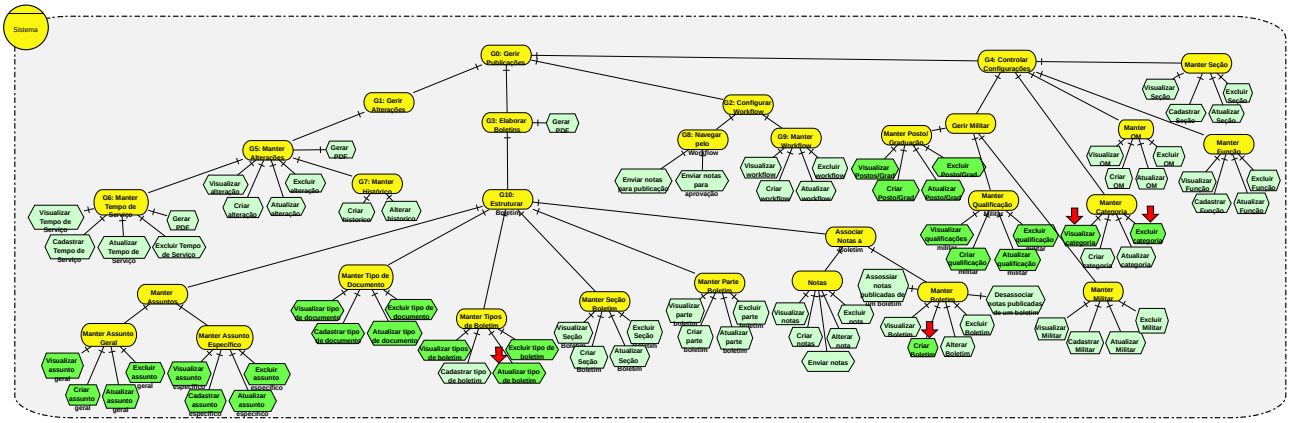
C.1.3 Árvore de Objetivos – Sprint 3



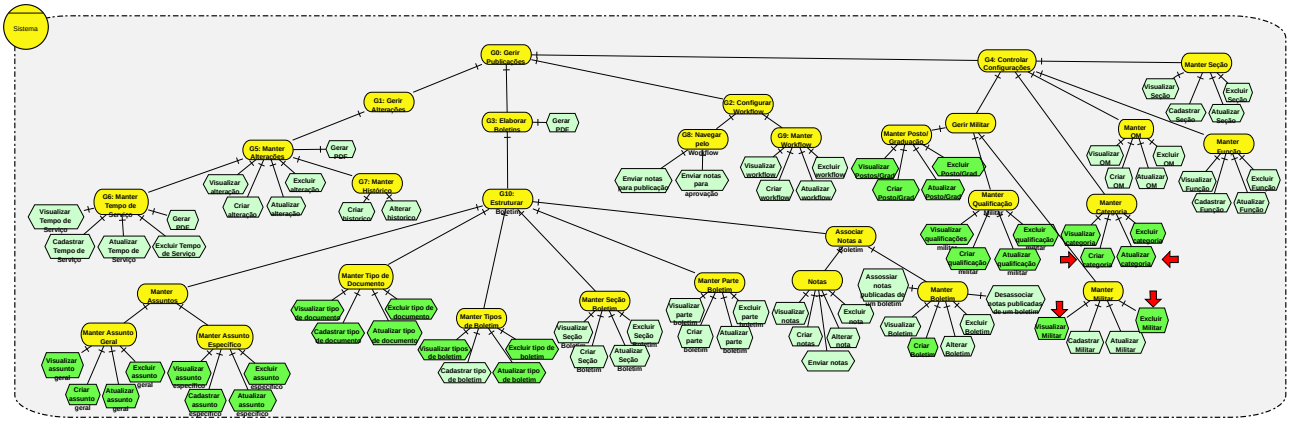
C.1.4 Árvore de Objetivos – Sprint 4



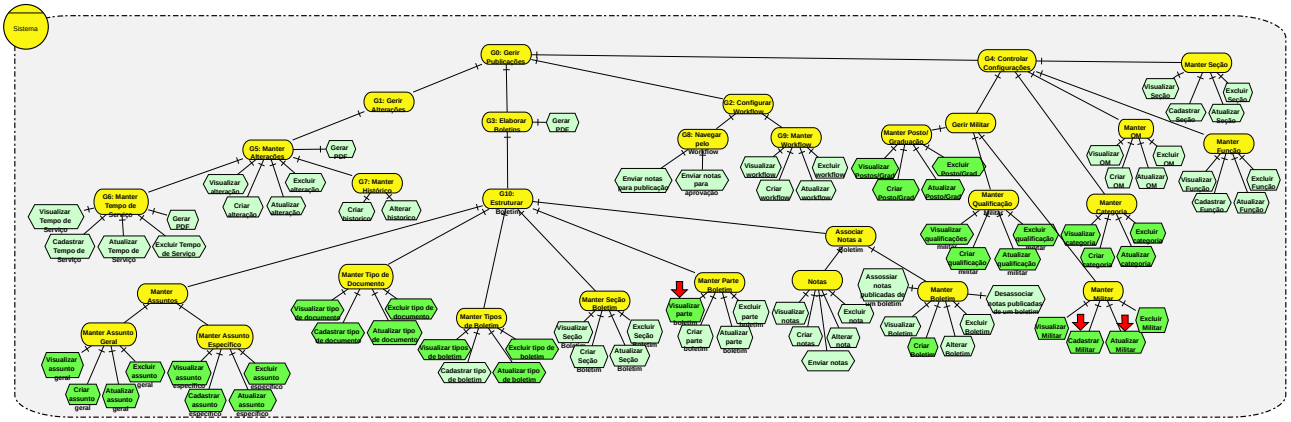
C.1.5 Árvore de Objetivos – Sprint 5



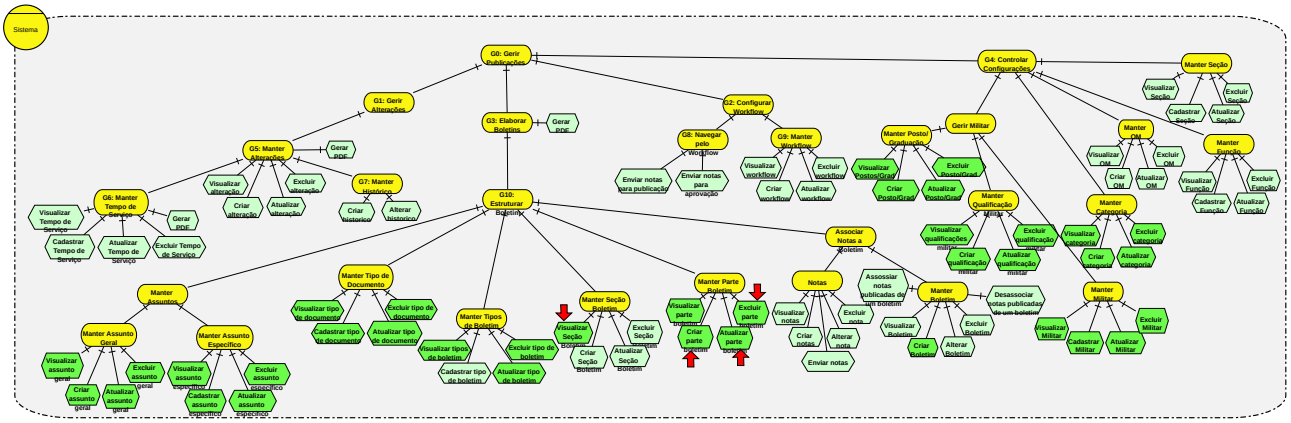
C.1.6 Árvore de Objetivos – Sprint 6



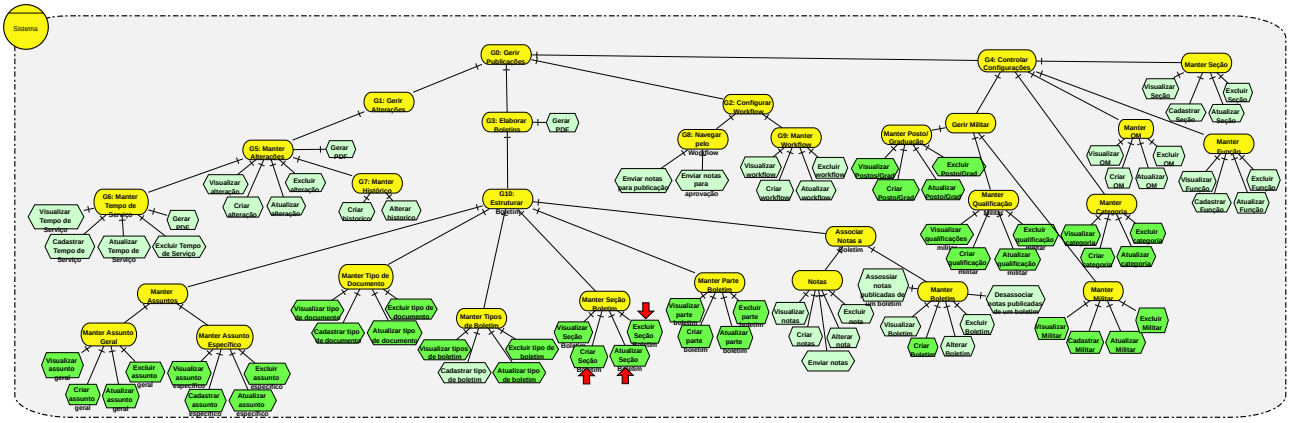
C.1.7 Árvore de Objetivos – Sprint 7



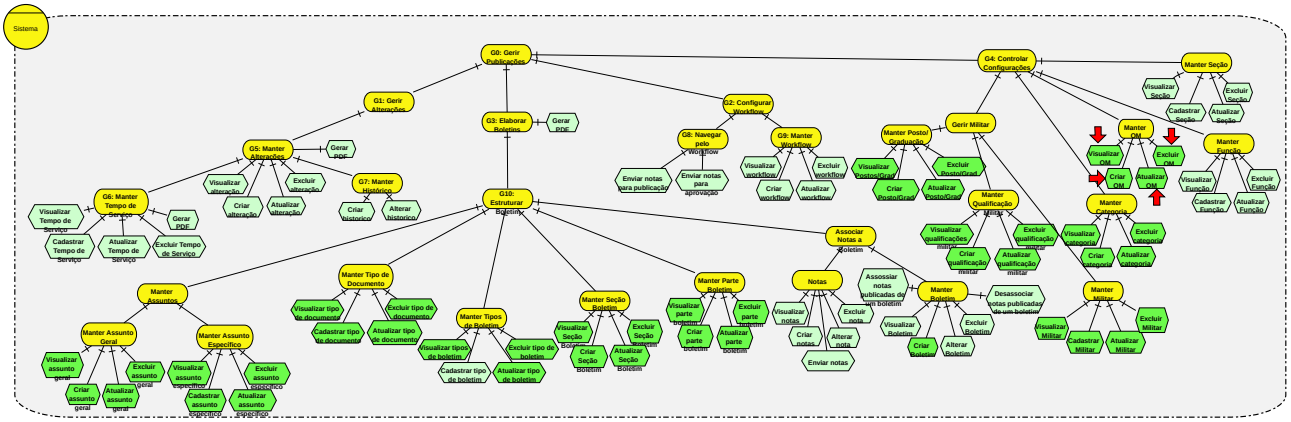
C.1.8 Árvore de Objetivos – Sprint 8



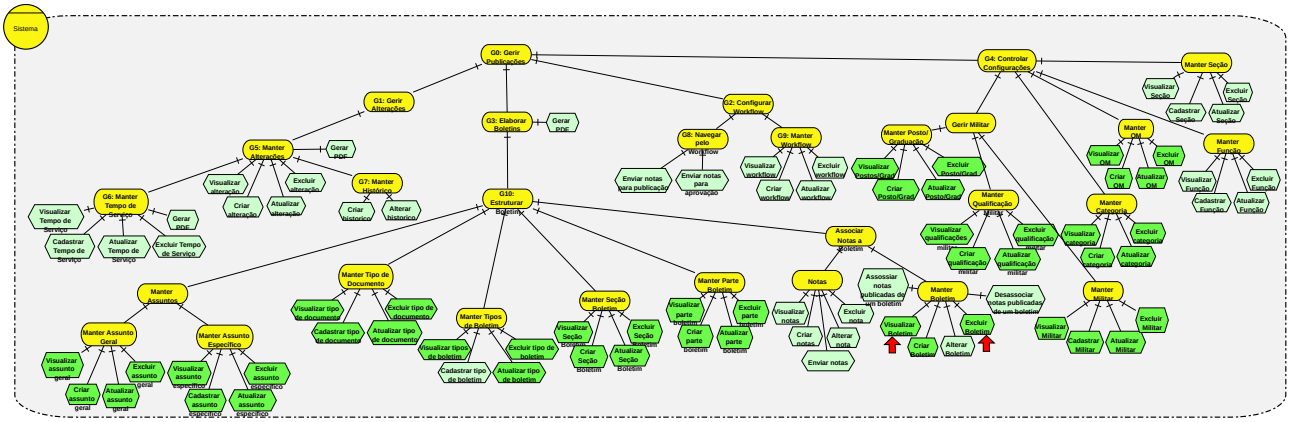
C.1.9 Árvore de Objetivos – Sprint 9



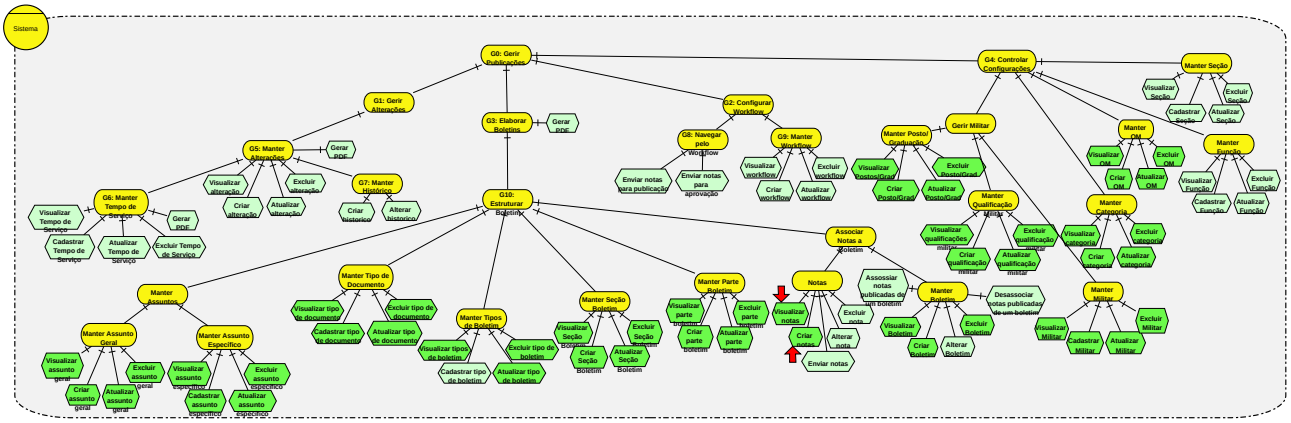
C.1.10 Árvore de Objetivos – Sprint 10



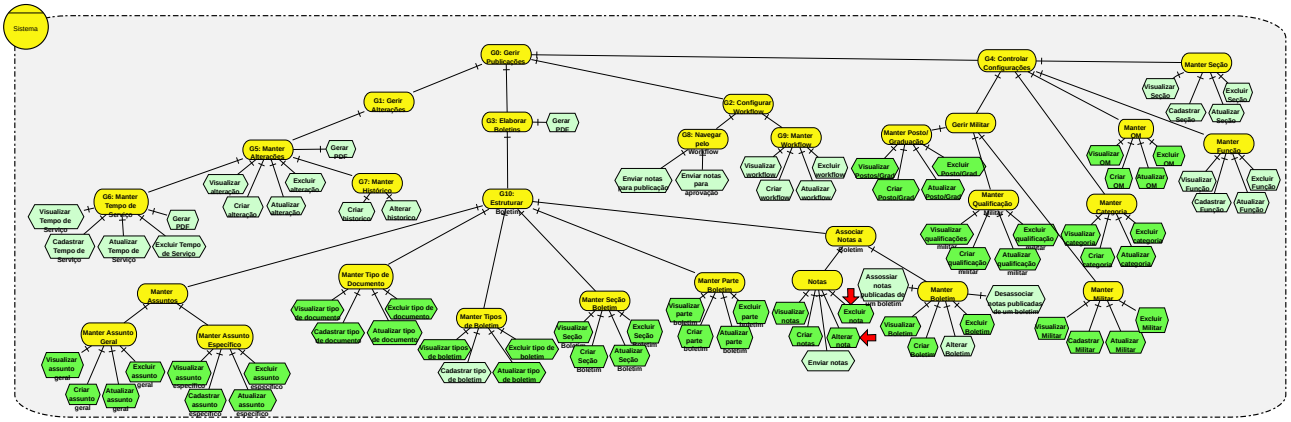
C.1.11 Árvore de Objetivos – Sprint 11



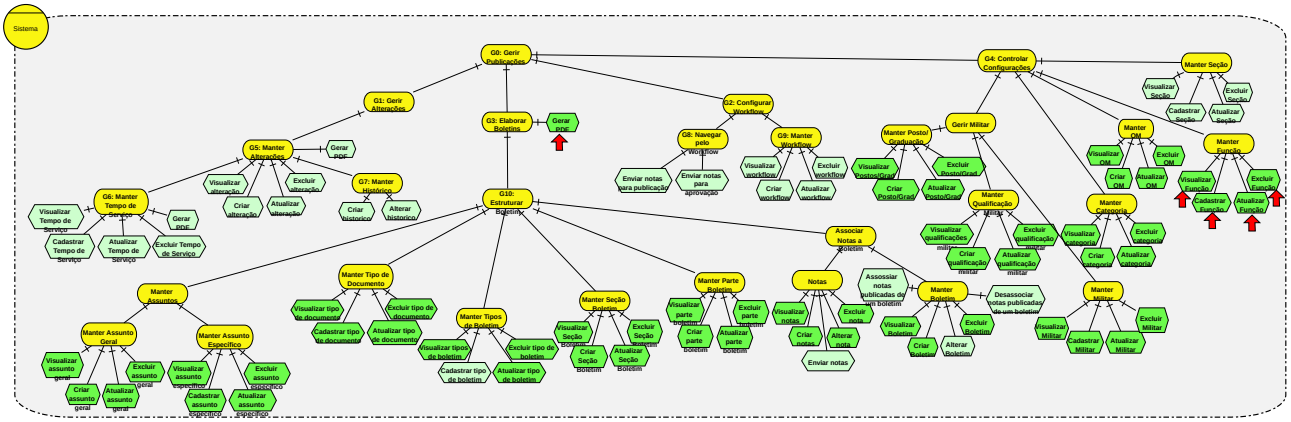
C.1.12 Árvore de Objetivos – Sprint 12



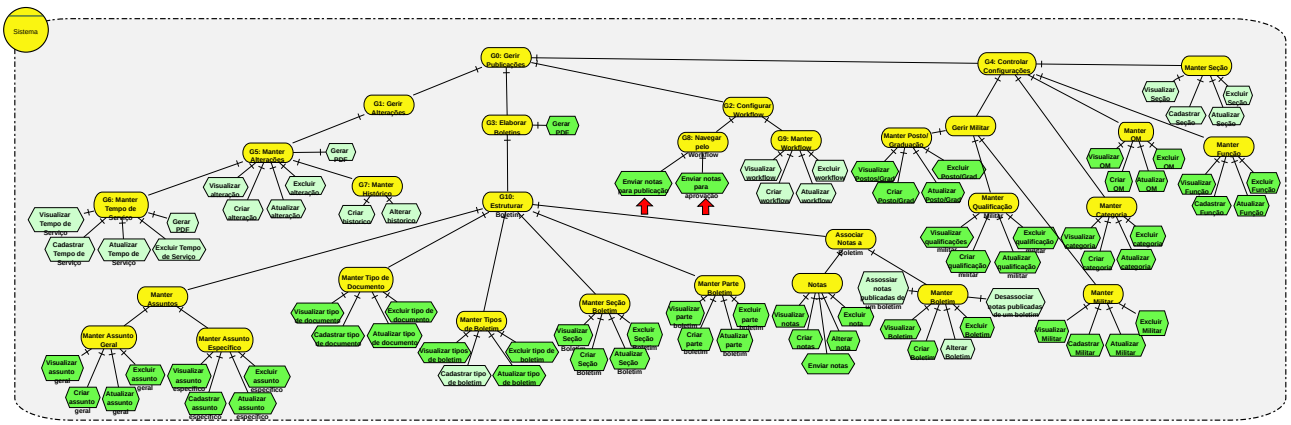
C.1.13 Árvore de Objetivos – Sprint 13



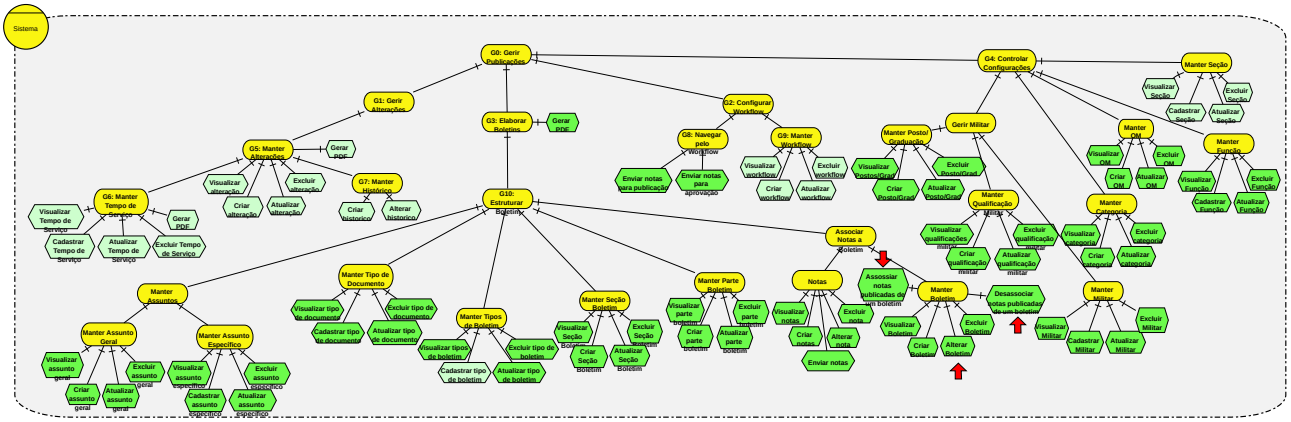
C.1.14 Árvore de Objetivos – Sprint 14



C.1.15 Árvore de Objetivos – Sprint 15



C.1.16 Árvore de Objetivos – Sprint 16

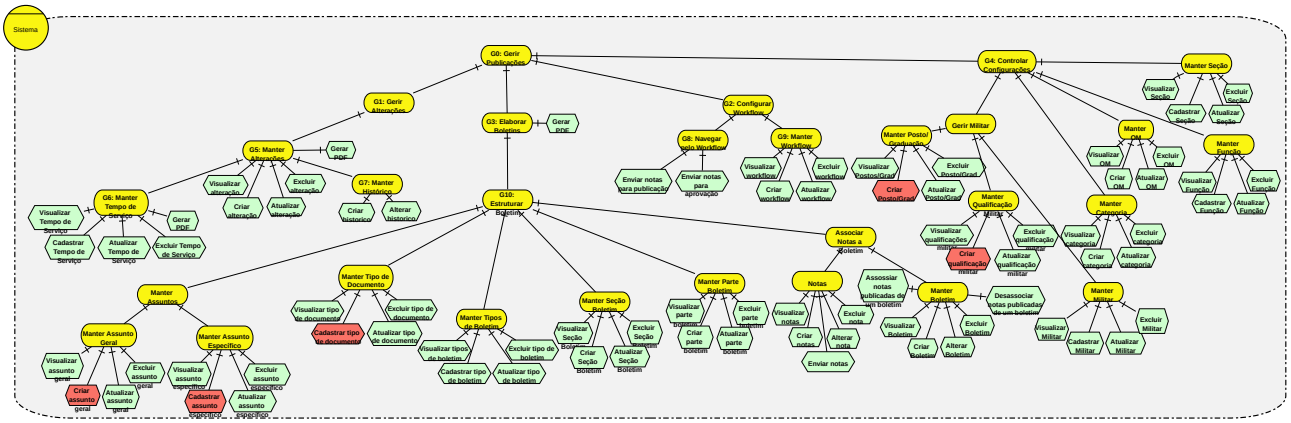


Apêndice D

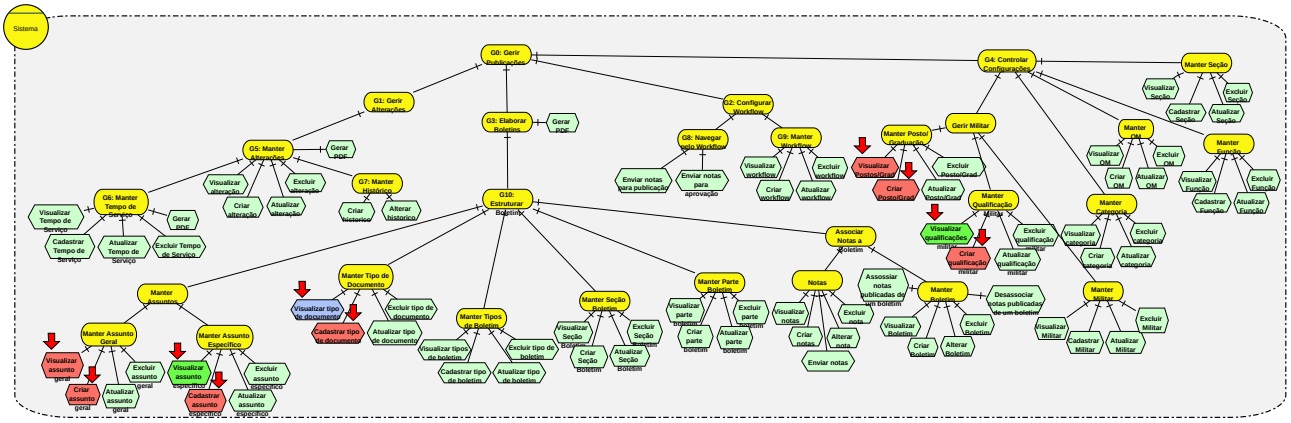
Evolução da Árvore de Objetivos no Projeto Mal-Sucedido

D.1 Evolução da Árvore de Objetivos

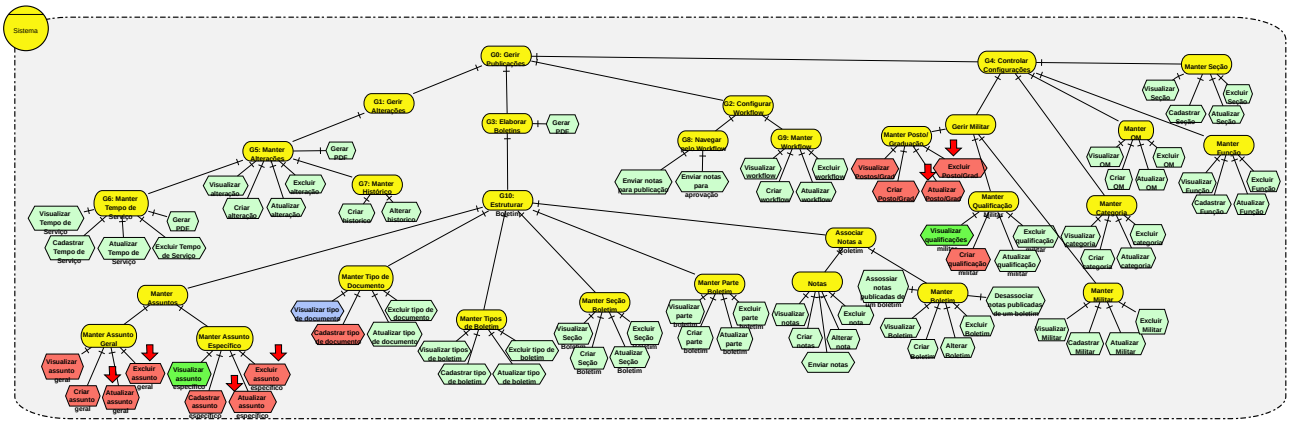
D.1.1 Árvore de Objetivos – Início do projeto



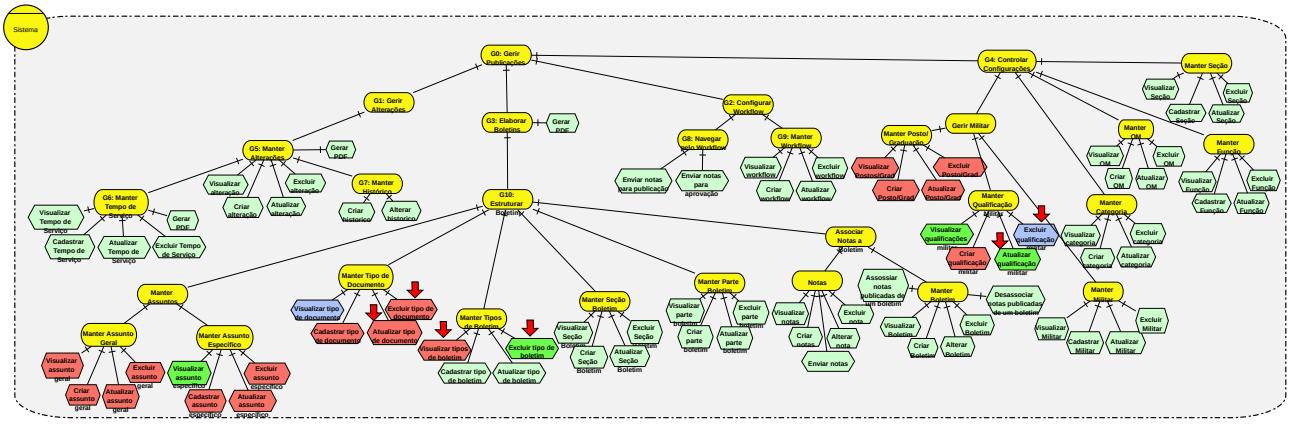
D.1.2 Árvore de Objetivos – Sprint 2



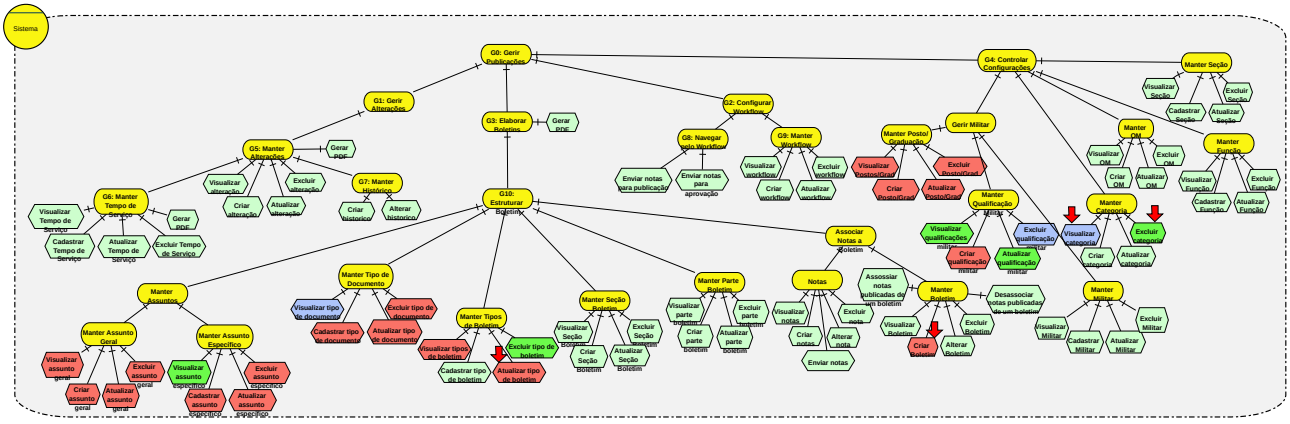
D.1.3 Árvore de Objetivos – Sprint 3



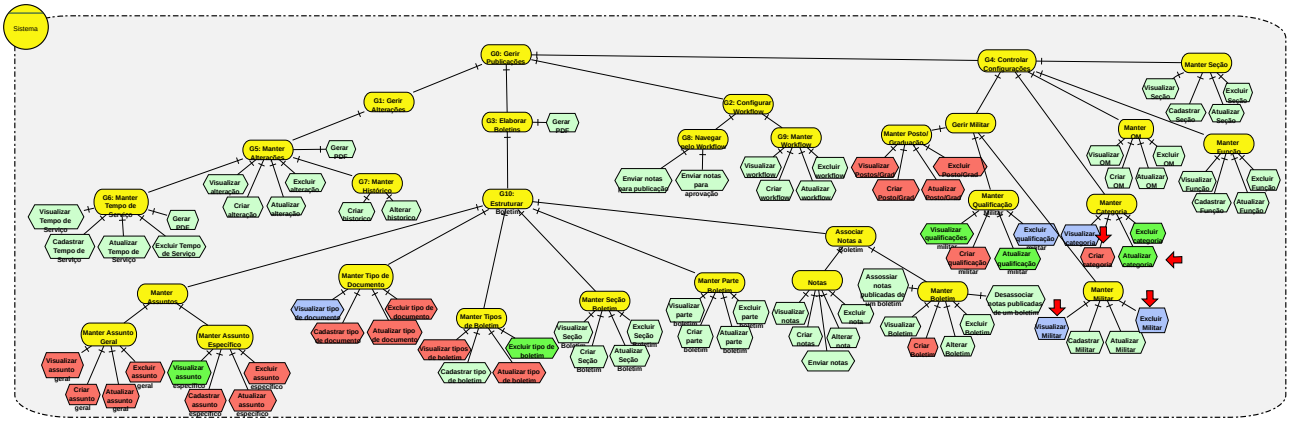
D.1.4 Árvore de Objetivos – Sprint 4



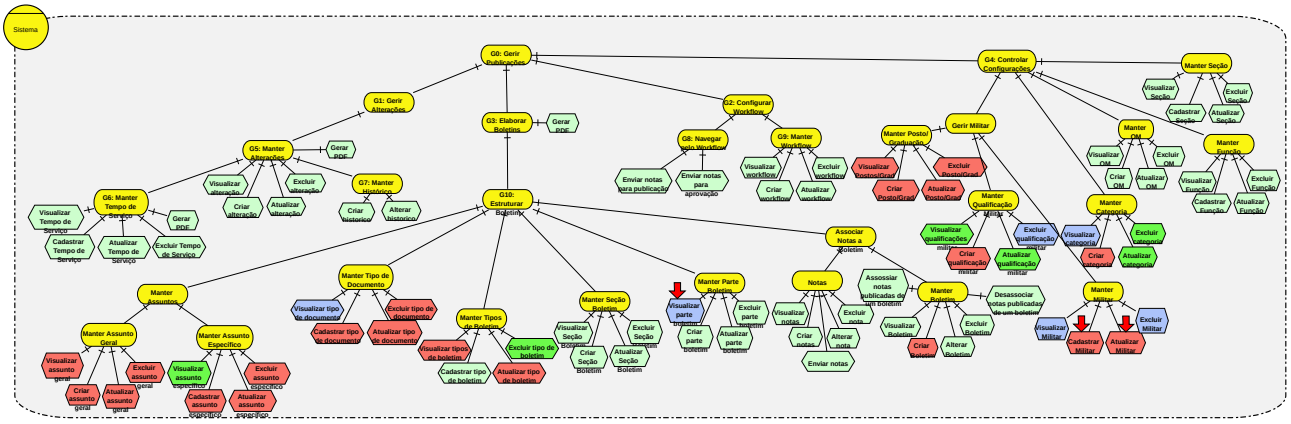
D.1.5 Árvore de Objetivos – Sprint 5



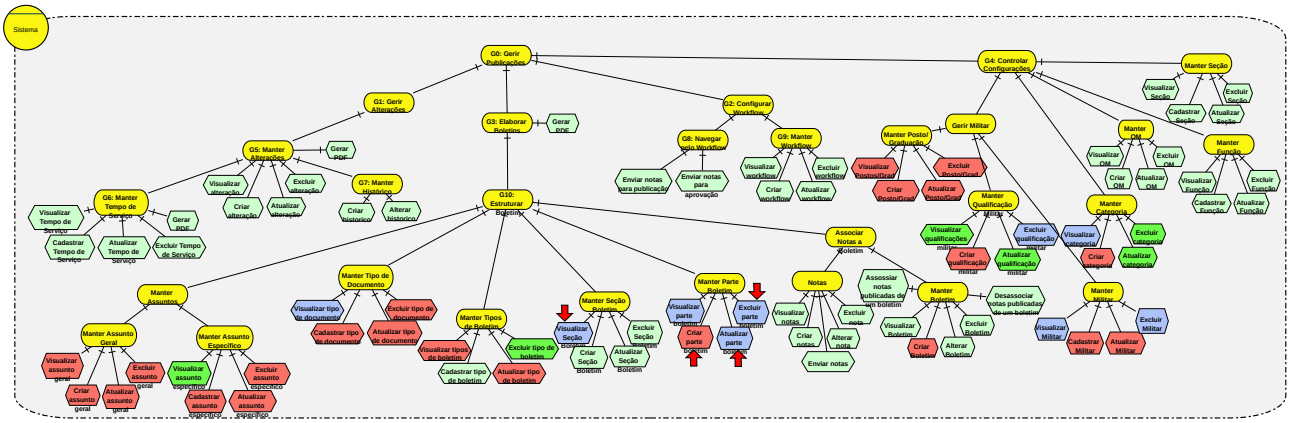
D.1.6 Árvore de Objetivos – Sprint 6



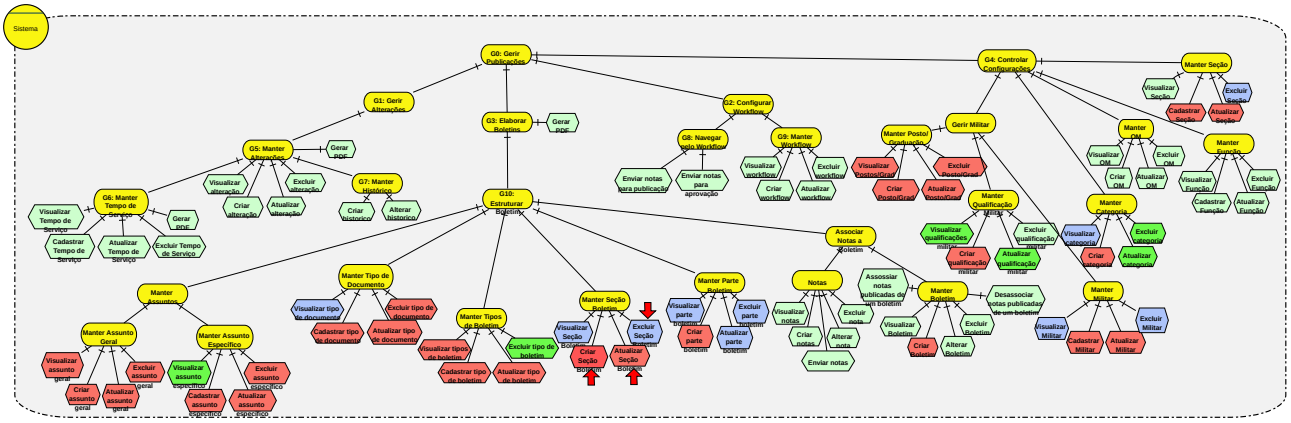
D.1.7 Árvore de Objetivos – Sprint 7



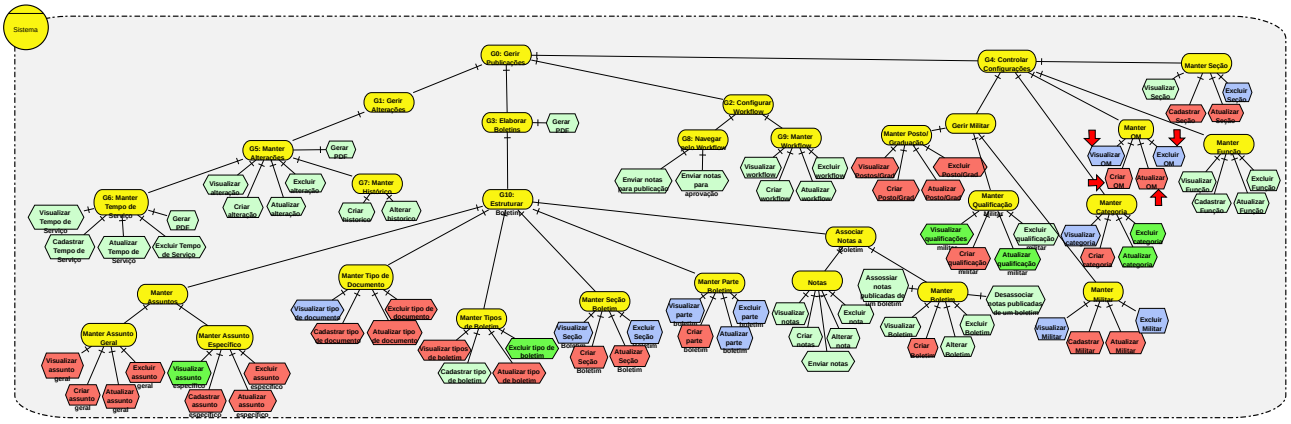
D.1.8 Árvore de Objetivos – Sprint 8



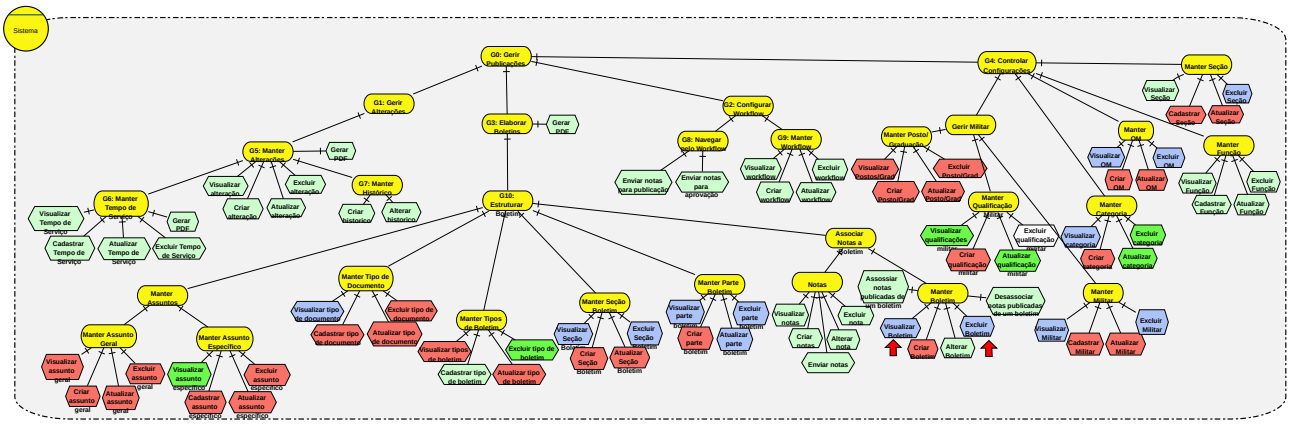
D.1.9 Árvore de Objetivos – Sprint 9



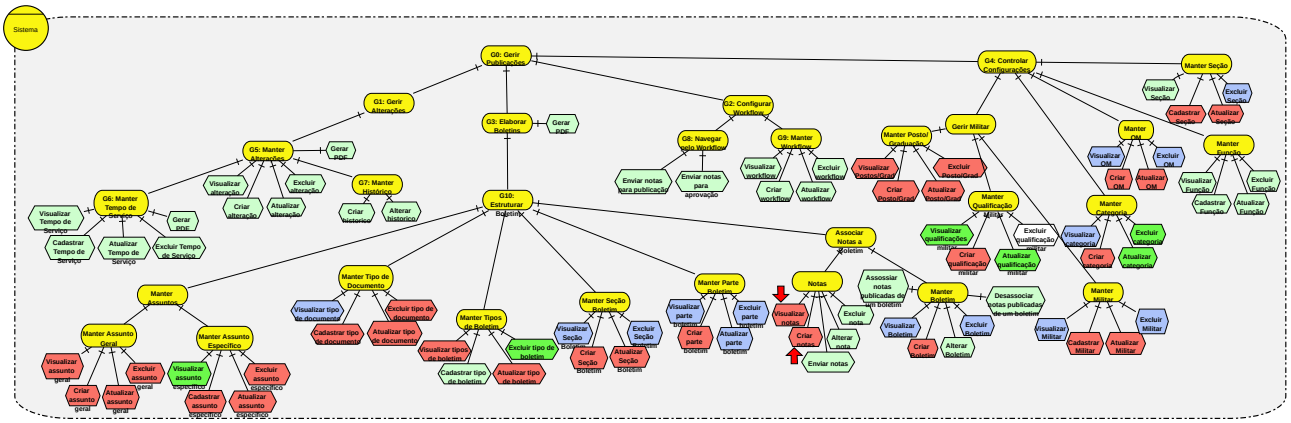
D.1.10 Árvore de Objetivos – Sprint 10



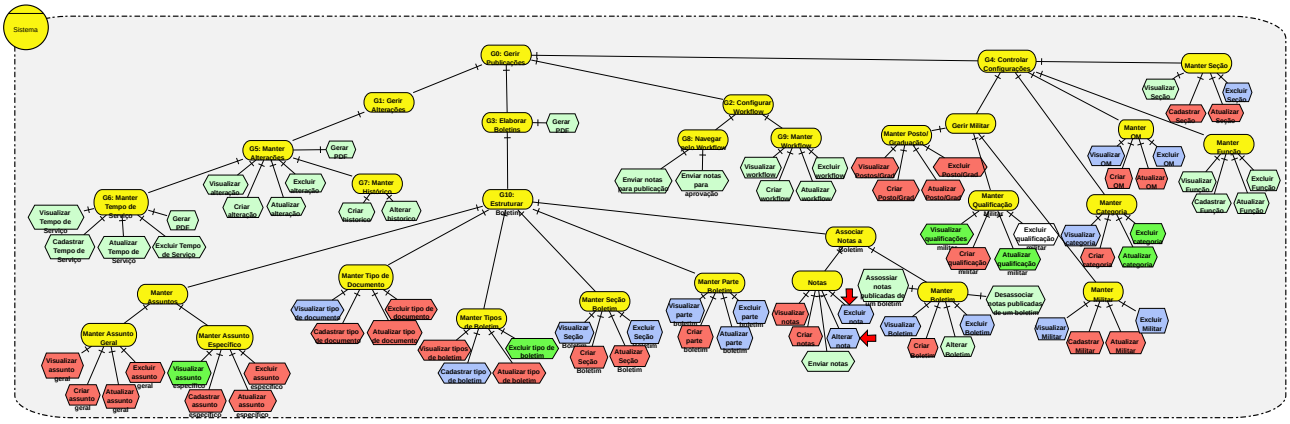
D.1.11 Árvore de Objetivos – Sprint 11



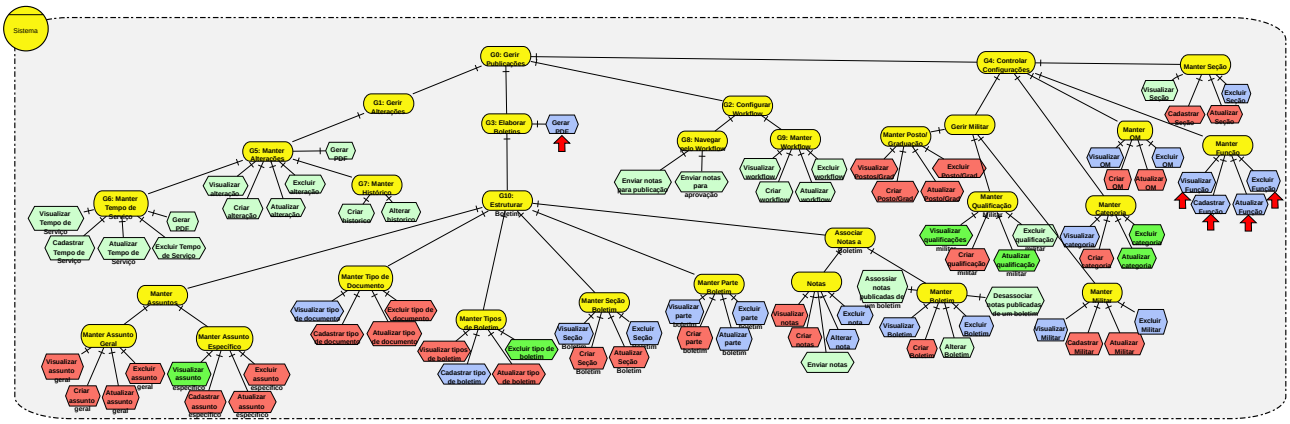
D.1.12 Árvore de Objetivos – Sprint 12



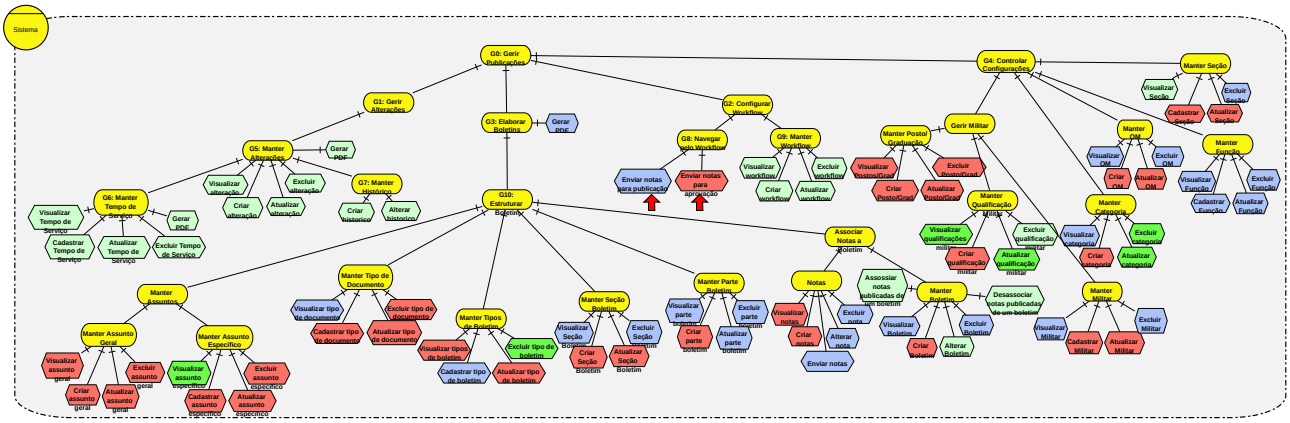
D.1.13 Árvore de Objetivos – Sprint 13



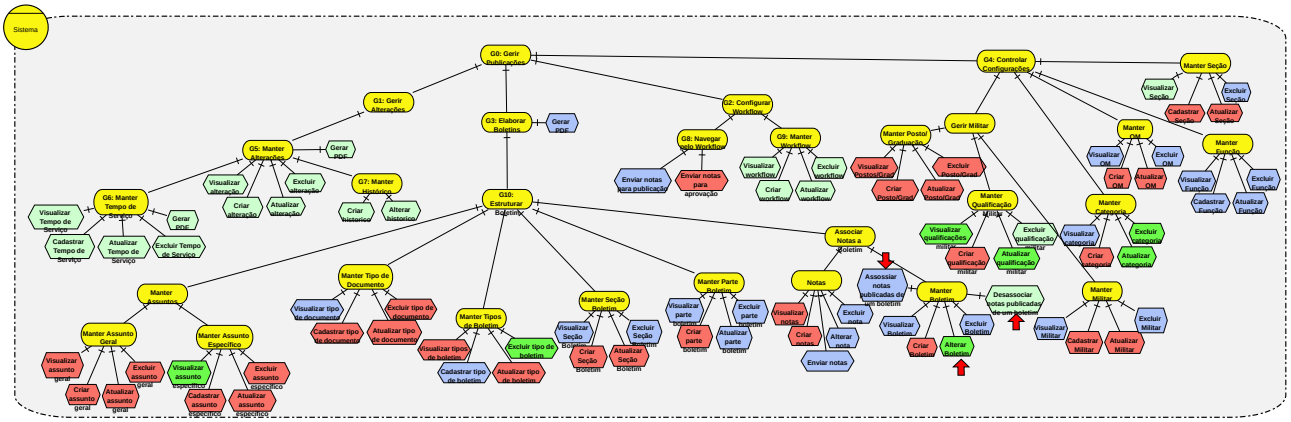
D.1.14 Árvore de Objetivos – Sprint 14



D.1.15 Árvore de Objetivos – Sprint 15



D.1.16 Árvore de Objetivos – Sprint 16



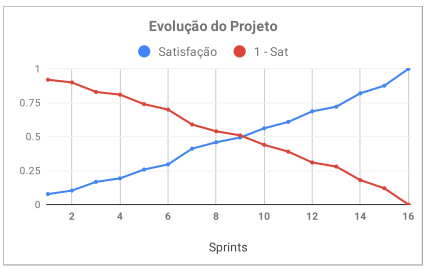
Apêndice E

Simulação de Evolução do Projeto

Configuração 1

Tipo	*Skiped*	*Fail*	*Sucess*
Peso	0	1	2

Sprint	Features	Complexidade	Cenários				Feature			
			Skiped	*Fail*	*Sucess*	Total Cenários	Status Atual	Status Ideal	Satisfação	1 - Sat
1	criar assunto especifico	1	0	0	2	2	4	4	1	0
	criar assunto geral	1	0	0	5	5	10	10	1	0
	criar posto/graduação	1	0	0	3	3	6	6	1	0
	criar tipo de documento	1	0	0	2	2	4	4	1	0
	criar qualificacao militar	1	0	0	6	6	12	12	1	0
2	visualizar assunto especifico	1	0	0	2	2	4	4	1	0
	visualizar assunto geral	1	0	0	1	1	2	2	1	0
	visualizar postos/graduações	1	0	0	1	1	2	2	1	0
	visualizar tipos de documento	1	0	0	1	1	2	2	1	0
	visualizar qualificações militar	1	0	0	1	1	2	2	1	0
3	excluir assunto especifico	1	0	0	2	2	4	4	1	0
	atualizar assunto especifico	1	0	0	2	2	4	4	1	0
	excluir assunto geral	1	0	0	2	2	4	4	1	0
	atualizar assunto geral	1	0	0	6	6	12	12	1	0
	excluir posto/graduação	1	0	0	1	1	2	2	1	0
	atualizar posto/graduação	1	0	0	2	2	4	4	1	0
4	excluir tipo de documento	1	0	0	1	1	2	2	1	0
	atualizar tipo de documento	1	0	0	1	1	2	2	1	0
	excluir qualificação militar	1	0	0	1	1	2	2	1	0
	atualizar qualificacao militar	1	0	0	1	1	2	2	1	0
	visualizar tipos de boletim	1	0	0	1	1	2	2	1	0
	excluir tipo de boletim	1	0	0	1	1	2	2	1	0
5	atualizar tipo de boletim	1	0	0	2	2	4	4	1	0
	criar tipo de boletim	1	0	0	9	9	18	18	1	0
	visualizar categoria	1	0	0	2	2	4	4	1	0
	excluir categoria	1	0	0	2	2	4	4	1	0
6	atualizar categoria	1	0	0	1	1	2	2	1	0
	criar categoria	1	0	0	4	4	8	8	1	0
	Visualizar Militar	1	0	0	2	2	4	4	1	0
	Excluir Militar	1	0	0	2	2	4	4	1	0
7	Atualizar Militar	1	0	0	11	11	22	22	1	0
	Cadastrar Militar	1	0	0	14	14	28	28	1	0
	visualizar parte boletim	1	0	0	2	2	4	4	1	0
8	excluir parte boletim	1	0	0	2	2	4	4	1	0
	atualizar parte boletim	1	0	0	2	2	4	4	1	0
	criar parte boletim	1	0	0	5	5	10	10	1	0
	visualizar secao	1	0	0	2	2	4	4	1	0
9	atualizar secao	1	0	0	3	3	6	6	1	0
	excluir secao	1	0	0	2	2	4	4	1	0
	cadastrar secao	1	0	0	3	3	6	6	1	0
10	visualizar organizacao militar	1	0	0	2	2	4	4	1	0
	excluir organizacao militar	1	0	0	2	2	4	4	1	0
	atualizar organizacao militar	1	0	0	6	6	12	12	1	0
	criar organizacao militar	1	0	0	6	6	12	12	1	0
11	visualizar secao boletim	1	0	0	2	2	4	4	1	0
	excluir secao boletim	1	0	0	2	2	4	4	1	0
	atualizar secao boletim	1	0	0	2	2	4	4	1	0
	criar secao boletim	1	0	0	5	5	10	10	1	0
12	visualizar notas	2	0	0	6	6	24	24	1	0
	criar notas para boletim	2	0	0	3	3	12	12	1	0
13	criar boletins	2	0	0	2	2	8	8	1	0
	excluir nota	1	0	0	2	2	4	4	1	0
	alterar notas	1	0	0	2	2	4	4	1	0
14	gerar pdf de nota	3	0	0	2	2	12	12	1	0
	visualizar funcao	1	0	0	2	2	4	4	1	0
	excluir funcao	1	0	0	2	2	4	4	1	0
	atualizar funcao	1	0	0	6	6	12	12	1	0
	cadastrar funcao	1	0	0	7	7	14	14	1	0
15	enviar notas para aprovação	3	0	0	1	1	6	6	1	0
	Enviar Notas para publicação	2	0	0	2	2	8	8	1	0
	Excluir Nota do Boletim	1	0	0	2	2	4	4	1	0
	Enviar Notas para o Boletim	2	0	0	2	2	8	8	1	0
16	excluir boletim	1	0	0	2	2	4	4	1	0
	alterar boletins	2	0	0	2	2	8	8	1	0
	associar notas publicadas boletim	5	0	0	4	4	40	40	1	0
	visualizar boletim	3	0	0	1	1	6	6	1	0
		82					**466**	**466**	**1.0**	**0.00**

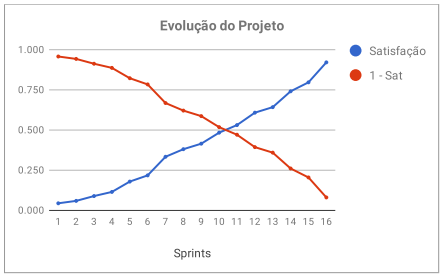


Evolução do Projeto				
Sprints	Atual	Ideal	Satisfação	1 - Sat
1	36	466	0.077	0.92
2	48	466	0.103	0.90
3	78	466	0.167	0.83
4	90	466	0.193	0.81
5	120	466	0.258	0.74
6	138	466	0.296	0.70
7	192	466	0.412	0.59
8	214	466	0.459	0.54
9	230	466	0.494	0.51
10	262	466	0.562	0.44
11	284	466	0.609	0.39
12	320	466	0.687	0.31
13	366	466	0.721	0.28
14	382	466	0.82	0.18
15	408	466	0.876	0.12
16	466	466	1	0

Configuração 2

Tipo	Skiped	Fail	Sucess
Peso	0	1	2

Sprint	Features	Complexidade	Cenários				Feature			
			Skiped	Fail	Sucess	Total Cenários	Status Atual	Status Ideal	Satisfação	1 - Sat
1	criar assunto especifico	1	0	1	1	2	3	4	0.75	0.25
	criar assunto geral	1	0	3	2	5	7	10	0.7	0.3
	criar posto/graduação	1	0	1	2	3	5	6	0.8333333333	0.167
	criar tipo de documento	1	1	1	0	2	1	4	0.25	0.75
	criar qualificacao militar	1	3	2	1	6	4	12	0.3333333333	0.666666666
2	visualizar assunto especifico	1	0	1	1	1	3	2	1.5	-0.5
	visualizar assunto geral	1	0	1	0	1	1	2	0.5	0.5
	visualizar postos/graduações	1	0	1	0	1	1	2	0.5	0.5
	visualizar tipos de documento	1	1	0	0	1	0	2	0	1
	visualizar qualificações militar	1	0	0	1	1	2	2	1	0
3	excluir assunto especifico	1	1	1	0	2	1	4	0.25	0.75
	atualizar assunto especifico	1	1	1	0	2	1	4	0.25	0.75
	excluir assunto geral	1	0	1	1	2	3	4	0.75	0.25
	atualizar assunto geral	1	2	3	1	6	5	12	0.4166666667	0.583333333
	excluir posto/graduação	1	0	1	0	1	1	2	0.5	0.5
	atualizar posto/graduação	1	0	1	1	2	3	4	0.75	0.25
4	excluir tipo de documento	1	0	0	1	1	2	2	1	0
	atualizar tipo de documento	1	0	0	1	1	2	2	1	0
	excluir qualificação militar	1	0	0	1	1	2	2	1	0
	atualizar qualificacao militar	1	0	0	1	1	2	2	1	0
	visualizar tipos de boletim	1	0	0	1	1	2	2	1	0
	excluir tipo de boletim	1	0	0	1	1	2	2	1	0
5	atualizar tipo de boletim	1	0	0	2	2	4	4	1	0
	criar tipo de boletim	1	0	0	9	9	18	18	1	0
	visualizar categoria	1	0	0	2	2	4	4	1	0
	excluir categoria	1	0	0	2	2	4	4	1	0
6	atualizar categoria	1	0	0	1	1	2	2	1	0
	criar categoria	1	0	0	4	4	8	8	1	0
	Visualizar Militar	1	0	0	2	2	4	4	1	0
	Excluir Militar	1	0	0	2	2	4	4	1	0
7	Atualizar Militar	1	0	0	11	11	22	22	1	0
	Cadastrar Militar	1	0	0	14	14	28	28	1	0
	visualizar parte boletim	1	0	0	2	2	4	4	1	0
8	excluir parte boletim	1	0	0	2	2	4	4	1	0
	atualizar parte boletim	1	0	0	2	2	4	4	1	0
	criar parte boletim	1	0	0	5	5	10	10	1	0
	visualizar secao	1	0	0	2	2	4	4	1	0
9	atualizar secao	1	0	0	3	3	6	6	1	0
	excluir secao	1	0	0	2	2	4	4	1	0
	cadastrar secao	1	0	0	3	3	6	6	1	0
10	visualizar organizacao militar	1	0	0	2	2	4	4	1	0
	excluir organizacao militar	1	0	0	2	2	4	4	1	0
	atualizar organizacao militar	1	0	0	6	6	12	12	1	0
	criar organizacao militar	1	0	0	6	6	12	12	1	0
11	visualizar secao boletim	1	0	0	2	2	4	4	1	0
	excluir secao boletim	1	0	0	2	2	4	4	1	0
	atualizar secao boletim	1	0	0	2	2	4	4	1	0
	criar secao boletim	1	0	0	5	5	10	10	1	0
12	visualizar notas	2	0	0	6	6	24	24	1	0
	criar notas para boletim	2	0	0	3	3	12	12	1	0
13	criar boletins	2	0	0	2	2	8	8	1	0
	excluir nota	1	0	0	2	2	4	4	1	0
	alterar notas	1	0	0	2	2	4	4	1	0
14	gerar pdf de nota	3	0	0	2	2	12	12	1	0
	visualizar funcao	1	0	0	2	2	4	4	1	0
	excluir funcao	1	0	0	2	2	4	4	1	0
	atualizar funcao	1	0	0	6	6	12	12	1	0
	cadastrar funcao	1	0	0	7	7	14	14	1	0
15	enviar notas para aprovação	3	0	0	1	1	6	6	1	0
	Enviar Notas para publicação	2	0	0	2	2	8	8	1	0
	Excluir Nota do Boletim	1	0	0	2	2	4	4	1	0
	Enviar Notas para o Boletim	2	0	0	2	2	8	8	1	0
16	excluir boletim	1	0	0	2	2	4	4	1	0
	alterar boletins	2	0	0	2	2	8	8	1	0
	associar notas publicadas boletim	5	0	0	4	4	40	40	1	0
	visualizar boletim	3	0	0	1	1	6	6	1	0
		82					**429**	**464**	**92.5**	**0.08**

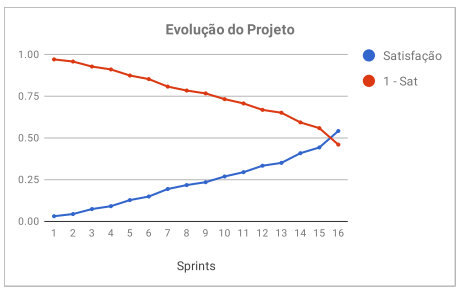


Evolução do Projeto				
Sprints	Atual	Ideal	Satisfação	1 - Sat
1	20	466	0.043	0.957
2	27	466	0.058	0.942
3	41	466	0.088	0.912
4	53	466	0.114	0.886
5	83	466	0.178	0.822
6	101	466	0.217	0.783
7	155	466	0.333	0.667
8	177	466	0.380	0.620
9	193	466	0.414	0.586
10	225	466	0.483	0.517
11	247	466	0.530	0.470
12	283	466	0.607	0.393
13	299	466	0.642	0.358
14	345	466	0.740	0.260
15	371	466	0.796	0.204
16	429	466	0.921	0.079

Configuração 3

Tipo	*Skiped*	*Fail*	*Sucess*
Peso	0	1	2

Sprint	Features	Complexidade	Cenários				Feature			
			Skiped	*Fail*	*Sucess*	Total Cenários	Status Atual	Status Ideal	Satisfação	1 - Sat
1	criar assunto especifico	1	0	1	1	2	3	4	0.75	0.25
	criar assunto geral	1	1	3	1	5	5	10	0.5	0.5
	criar posto/graduação	1	2	1	0	3	1	6	0.167	0.833
	criar tipo de documento	1	1	1	0	2	1	4	0.25	0.75
	criar qualificacao militar	1	3	2	1	6	4	12	0.333	0.667
2	visualizar assunto especifico	1	0	0	1	1	2	2	1	0
	visualizar assunto geral	1	0	1	0	1	1	2	0.5	0.5
	visualizar postos/graduações	1	0	1	0	1	1	2	0.5	0.5
	visualizar tipos de documento	1	1	0	0	1	0	2	0	1
	visualizar qualificações militar	1	0	0	1	1	2	2	1	0
3	excluir assunto especifico	1	1	1	0	2	1	4	0.25	0.75
	atualizar assunto especifico	1	1	1	0	2	1	4	0.25	0.75
	excluir assunto geral	1	0	1	1	2	3	4	0.75	0.25
	atualizar assunto geral	1	2	3	1	6	5	12	0.417	0.583
	excluir posto/graduação	1	0	1	0	1	1	2	0.5	0.5
	atualizar posto/graduação	1	0	1	1	2	3	4	0.75	0.25
4	excluir tipo de documento	1	0	1	0	1	1	2	0.5	0.5
	atualizar tipo de documento	1	0	0	1	1	2	2	1	0
	excluir qualificação militar	1	1	0	0	1	0	2	0	1
	atualizar qualificacao militar	1	0	0	1	1	2	2	1	0
	visualizar tipos de boletim	1	0	1	0	1	1	2	0.5	0.5
	excluir tipo de boletim	1	0	0	1	1	2	2	1	0
5	atualizar tipo de boletim	1	0	1	1	2	3	4	0.75	0.25
	criar tipo de boletim	1	2	3	4	9	11	18	0.611	0.389
	visualizar categoria	1	2	0	0	2	0	4	0	1
	excluir categoria	1	0	1	1	2	3	4	0.75	0.25
6	atualizar categoria	1	0	0	1	1	2	2	1	0
	criar categoria	1	0	2	2	4	6	8	0.75	0.25
	Visualizar Militar	1	1	0	1	2	2	4	0.5	0.5
	Excluir Militar	1	2	0	0	2	0	4	0	1
7	Atualizar Militar	1	6	2	3	11	8	22	0.364	0.636
	Cadastrar Militar	1	7	3	4	14	11	28	0.393	0.607
	visualizar parte boletim	1	1	0	1	2	2	4	0.5	0.5
8	excluir parte boletim	1	1	0	1	2	2	4	0.5	0.5
	atualizar parte boletim	1	1	0	1	2	2	4	0.5	0.5
	criar parte boletim	1	2	1	2	5	5	10	0.5	0.5
	visualizar secao	1	1	0	1	2	2	4	0.5	0.5
9	atualizar secao	1	1	1	1	3	3	6	0.5	0.5
	excluir secao	1	1	0	1	2	2	4	0.5	0.5
	cadastrar secao	1	1	1	1	3	3	6	0.5	0.5
10	visualizar organizacao militar	1	1	0	1	2	2	4	0.5	0.5
	excluir organizacao militar	1	1	0	1	2	2	4	0.5	0.5
	atualizar organizacao militar	1	2	2	2	6	6	12	0.5	0.5
	criar organizacao militar	1	2	2	2	6	6	12	0.5	0.5
11	visualizar secao boletim	1	1	0	1	2	2	4	0.5	0.5
	excluir secao boletim	1	1	0	1	2	2	4	0.5	0.5
	atualizar secao boletim	1	0	1	1	2	3	4	0.75	0.25
	criar secao boletim	1	2	1	2	5	5	10	0.5	0.5
12	visualizar notas	2	2	2	2	6	12	24	0.5	0.5
	criar notas para boletim	2	1	1	1	3	6	12	0.5	0.5
13	criar boletins	2	1	0	1	2	4	8	0.5	0.5
	excluir nota	1	1	0	1	2	2	4	0.5	0.5
	alterar notas	1	1	0	1	2	2	4	0.5	0.5
14	gerar pdf de nota	3	1	0	1	2	6	12	0.5	0.5
	visualizar funcao	1	1	0	1	2	2	4	0.5	0.5
	excluir funcao	1	1	0	1	2	2	4	0.5	0.5
	atualizar funcao	1	2	0	4	6	8	12	0.667	0.333
	cadastrar funcao	1	2	1	4	7	9	14	0.643	0.357
15	enviar notas para aprovação	3	0	0	1	1	6	6	1	0
	Enviar Notas para publicação	2	1	0	1	2	4	8	0.5	0.5
	Excluir Nota do Boletim	1	1	0	1	2	2	4	0.5	0.5
	Enviar Notas para o Boletim	2	1	0	1	2	4	8	0.5	0.5
16	excluir boletim	1	1	0	1	2	2	4	0.5	0.5
	alterar boletins	2	0	0	2	2	8	8	1	0
	associar notas publicadas boletim	5	1	0	3	4	30	40	0.75	0.25
	visualizar boletim	3	0	0	1	1	6	6	1	0
		82					**252**	**464**	**54.3**	**0.46**



Evolução do Projeto				
Sprints	Atual	Ideal	Satisfação	1 - Sat
1	14	466	0.030	0.970
2	20	466	0.043	0.957
3	34	466	0.073	0.927
4	42	466	0.090	0.910
5	59	466	0.127	0.873
6	69	466	0.148	0.852
7	90	466	0.193	0.807
8	101	466	0.217	0.783
9	109	466	0.234	0.766
10	125	466	0.268	0.732
11	137	466	0.294	0.706
12	155	466	0.333	0.667
13	163	466	0.350	0.650
14	190	466	0.408	0.592
15	206	466	0.442	0.558
16	252	466	0.541	0.459

Apêndice F

Programa R: Scripts de plotagens

```
#*********
#SCRIPT 1
#*********
#------------------------------
# CÁLCULO DA SATISFAÇÃO DAS FEATURES
# plot baseado nos dados Configuração 1
# Pesquisa de Mestrado UNB
# Autor: Fábio Barros Leal
#------------------------------
# Orientadora: Dra. Genaína Nunes Rodrigues

# Limpa plots
if(!is.null(dev.list())) dev.off()

# Limpa console
cat("\014")

# Limpa workspace
rm(list=ls())

# Configura o diretório de trabalho
# setwd("/home/z/Devel/zero/lealfb/bdd2Goal/R")

matriz <- read.csv(file="matriz.csv", header=FALSE, sep=",")

#print(matriz)
```

```
peso_skipped <- 0
peso_fail <- 1
peso_success <- 2

colnames(matriz) <- c('Sprint', 'Features', 'Complexidade', 'Skipped', 'Fail', 'Succe
  'Total Cenários', 'Status Atual', 'Status Ideal', 'Satisfação', 'DT Local')

# matriz <- as.data.frame(t(matriz))

total_complexidade <- 0
total_status_atual <- 0
total_status_ideal <- 0
total_satisfacao <- 0
total_dt <- 0

maximo_sprints <- 0

for ( i in 1:nrow(matriz) ) {
    calc_skipped <- as.numeric(matriz[i,4]) * peso_skipped
        calc_fail <- as.numeric(matriz[i,5]) * peso_fail
        calc_success <- as.numeric(matriz[i,6]) * peso_success
        complexidade <- as.numeric(matriz[i,3])

        total_cenarios <- as.numeric(matriz[i,7])

        status_atual <- (calc_skipped + calc_fail + calc_success) * complexidade
        status_ideal <- (total_cenarios * peso_success) * complexidade
        satisfacao <- (status_atual*100)/status_ideal
        dt_local <- 1 - (status_atual/status_ideal)

        matriz[i,8] <- status_atual
        matriz[i,9] <- status_ideal
        matriz[i,10] <- satisfacao
        matriz[i,11] <- dt_local

        total_complexidade <- complexidade + total_complexidade
        total_status_atual <- status_atual + total_status_atual
```

```
        total_status_ideal <- status_ideal + total_status_ideal

        if (as.integer(matriz[i,1]) > maximo_sprints) {
                maximo_sprints <- as.integer(matriz[i,1])
        }
}

total_satisfacao <- total_status_atual / total_status_ideal
total_dt <- 1 - (total_status_atual/total_status_ideal)

evolucao_dt <- rbind()

for ( i in 1:maximo_sprints ) {
        atual <- 0

        for ( j in 1:nrow(matriz) ) {
                if (as.integer(matriz[j,1]) > i) {
                        break
                }
                atual <- atual + as.integer(matriz[j,8])
        }

        sprint <- c(i, atual, as.integer(total_status_ideal),
            (atual/total_status_ideal),
                1-(atual/total_status_ideal)
        )
        evolucao_dt <- rbind(evolucao_dt, sprint[1:5])
}

evolucao_dt <- as.data.frame(evolucao_dt)

colnames(evolucao_dt) <- c('Sprint', 'Atual', 'Ideal', 'Satisfação', 'DT Global')

#svg(filename="grafico_1.svg", width=5, height=4, pointsize=12)

plot(evolucao_dt[,1], evolucao_dt[,4], type = "b", pch = 15, col = "blue", xlab = "Spr
lines(evolucao_dt[,1], evolucao_dt[,5], type = "b", pch = 19, col = "red", xlab = "Spr
```

```
legend("top", legend=c("Satisfação", "DT Global"), col=c("blue", "red"), lty = 1:1, c
axis(side=1, at=0:maximo_sprints, labels=0:maximo_sprints)

save.image()

#dev.off()
```