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

Uma Proposta para Priorização de Testes Funcionais Orientada a Modelo de Objetivos

Nayara Rossi Brito da Silva

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da
Computação

Orientadora
Prof.a Dr.a Genaina Nunes Rodrigues

Brasília
2022

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

Uma Proposta para Priorização de Testes Funcionais Orientada a Modelo de Objetivos

Nayara Rossi Brito da Silva

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da
Computação

Prof.a Dr.a Genaina Nunes Rodrigues (Orientadora)
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 17 de maio de 2022

Uma Proposta para Priorização de Testes Funcionais Orientada a Modelo de Objetivos

Nayara Rossi Brito da Silva*
nayararossisilva@gmail.com
Universidade de Brasília
Brasília, Distrito Federal, Brasil

Genaina Nunes Rodrigues*
genaina@unb.br
Universidade de Brasília
Brasília, Distrito Federal, Brasil

RESUMO

O método ágil permite que modificações de features sejam feitas ao longo do processo de desenvolvimento, porém quando uma feature é modificada, é necessário verificar o funcionamento do sistema através do processo de testagem. Isso aumenta o número de testes no projeto, aumentando o custo de desenvolvimento e, por vezes, dificultando a execução de toda a suíte de testes. Nesse cenário, a priorização de testes torna-se um importante meio de diminuir esse custo, porém surge a questão de como realizar essa priorização. Além disso, no método ágil as entregas são feitas em iterações de curta duração e nem sempre é possível entregar tudo o que foi planejado. Nessas ocasiões é importante diminuir a insatisfação dos stakeholders através da entrega de requisitos prioritários para os mesmos. Considerando essas duas situações, esse trabalho busca auxiliar os times de desenvolvimento na priorização de testes utilizando como base os requisitos mais prioritários para o cliente. Para isso, propõe-se um modelo de priorização de testes utilizando a abordagem BDD-GORE. O trabalho foi avaliado através de um questionário de uso baseado em perguntas do System Usability Scale em conjunto com o teste prático da ferramenta realizado por dois desenvolvedores voluntários. Os resultados preliminares indicam que as prioridades sugeridas pela ferramenta cumprem seu papel de auxiliar o gerenciamento do projeto, apesar de possuir espaço para melhorias.

CCS CONCEPTS

• **Software and It's Engineering**; • **Software Creation and Management**; • **Software Process Management**; • **Software Development Methods** → Agile Software Development;

KEYWORDS

TCP, Priorização de Testes, BDD, GORE

ACM Reference Format:
Nayara Rossi Brito da Silva and Genaina Nunes Rodrigues. 2022. Uma Proposta para Priorização de Testes Funcionais Orientada a Modelo de Objetivos. In *Proceedings of Maio, 2022 (Trabalho de Graduação).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Trabalho de Graduação, Brasília, DF,

https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Quando há uma mudança em um software, ela pode causar erros não apenas na funcionalidade em que ocorreu, mas em outras partes ligadas à essa funcionalidade. Para ter certeza que isso não vai acontecer, são utilizados alguns métodos como: testes de regressão e desenvolvimento baseado em testes.

Testes de regressão são testes criados para verificar o funcionamento de um código já existente [10]. Já o desenvolvimento baseado em testes realiza primeiro a implementação do teste para depois desenvolver o código. Logo, se há alguma modificação no requisito do software essas mudanças devem se refletir primeiro nos testes a serem criados e depois no código, o que faz com que os testes estejam sempre acompanhando as modificações.

No entanto, mesmo considerando as diretrizes dessa perspectiva, mudanças constantes no software aumentam o número de testes, o que encarece o processo de desenvolvimento [27]. Em um projeto real, com limitações de recursos como tempo, orçamento e pessoal, isso dificulta a execução de toda a suíte de testes[3]. Em função desse cenário surgem discussões sobre quais testes seriam prioritários e como definir essa priorização [22].

Uma das técnicas utilizadas para isso é a Test Case Prioritization, ou TCP [27], que é um método de ordenação de testes que tem como objetivo maximizar seus benefícios para o desenvolvedor. Assim, os testes são priorizados de acordo com algum critério, como: importância para o cliente; complexidade; possibilidade de erro da funcionalidade; entre outros. Logo, os testes mais importantes são implementados primeiro, garantindo que os pontos prioritários do sistema já tenham sido trabalhados. Desta forma, esses pontos já estariam funcionando corretamente na entrega caso não seja possível executar todos os testes de determinada versão do sistema.

Isso se mostra útil especialmente no contexto de métodos ágeis, onde as entregas são feitas em iterações curtas. Como nem sempre é possível incluir tudo o que foi planejado em uma entrega, AbdElazim et al. [3] comentam que é importante diminuir a insatisfação do cliente.

Essa insatisfação pode ser reduzida com a entrega das funcionalidades do software que os stakeholders considerem mais cruciais [3]. Portanto, é importante priorizar a implementação dessas funcionalidades. Ao utilizar o desenvolvimento guiado a testes é possível priorizar os testes funcionais relativos às features mais relevantes, isso é o chamado *Requirements Test Prioritization* [21] ou priorização de testes através de requisitos.

Outro ponto importante da priorização de testes por requisitos é que ela permite encontrar os erros que mais afetam o sistema. Isso é interessante pois sistemas podem possuir funcionalidades principais e secundárias. Assim, é mais importante identificar um erro que afete uma funcionalidade principal, pois esse tipo de erro

tem mais chance de impactar negativamente o uso do software e incomodar mais o usuário [19].

Como exemplo, temos muitos jogos que permitem que o jogador confira a hora em um relógio na tela, apesar disso, essa não é sua funcionalidade principal. Assim, se essa feature apresentar um erro, esse erro dentro do jogo seria menos crítico do que se um aplicativo de despertador apresentasse o mesmo defeito.

A priorização de testes baseada em requisitos mostra-se uma técnica interessante, porém complexa. Como nota Sachdeva et al. [28] a ordenação de testes fica mais difícil conforme o número de testes e requisitos aumenta. Além disso, diversos fatores podem ser considerados na hora da priorização como: complexidade da funcionalidade, preferência dos *stakeholders* (clientes, *project owners* e outras pessoas interessadas em participar do processo de desenvolvimento do projeto), propagação de erro, entre outros; e nem sempre é fácil saber qual utilizar. Torna-se necessário então fornecer ferramentas para a equipe de desenvolvimento que facilitem essas decisões.

Assim, esse trabalho propõe desenvolver uma extensão de uma ferramenta chamada BDD-GORE [17] para auxiliar o time de desenvolvimento na priorização de testes funcionais. Essa priorização será feita baseada nos requisitos do projeto e considerando o contexto de desenvolvimento ágil com a utilização do Desenvolvimento Orientado a Comportamento, ou *Behavior Driven Development* (BDD).

O BDD é um método que combina testes de aceitação com a elicitação de requisitos e é utilizado desde o início do desenvolvimento do projeto. A ferramenta BDD-GORE utiliza BDD junto com a modelagem de objetivos (goal modelling) para melhorar o processo de documentação viva do projeto, permitindo também a visualização de quais testes no sistema já sucederam ou falharam.

Essa visualização é possível devido à relação que existe entre testes e requisitos dentro do método do BDD. Em função do BDD-GORE oferecer essa visualização decidiu-se utilizar a abordagem de *Requirements Test Prioritization* para aplicar técnicas de priorização de testes (TCP) nessa ferramenta.

Tal extensão oferece como contribuição ao processo de desenvolvimento ágil um mecanismo que auxilia a equipe de desenvolvimento a gerenciar a priorização de testes. Adicionalmente, a equipe de desenvolvimento pode utilizar esse mecanismo para facilitar a definição de mínimos produtos viáveis (MVP - minimum viable product) que possuam maior qualidade. Assim, busca-se diminuir a insatisfação do cliente no caso da entrega por etapas (sprints), situação bastante comum em projetos de desenvolvimento de software.

Para entendimento do desenvolvimento deste trabalho, as próximas seções estão organizadas da seguinte forma: a seção 2 apresenta a fundamentação teórica e discorre sobre os métodos escolhidos para a priorização de testes e sobre as ferramentas mencionadas; a seção 3 explica a solução em sua forma final e o processo de testagem da ferramenta com o público alvo; a seção 4 apresenta os resultados e as impressões sobre o teste da ferramenta; a seção 5 discute alguns trabalhos relacionados e explica o processo de pesquisa que levou à construção da ferramenta; e a seção 6 apresenta conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Aqui será abordado um breve entendimento sobre Behaviour Driven Development e Goal Oriented Requirements Engineering que são as técnicas sobre as quais este trabalho foi pensado e desenvolvido. A ferramenta BDD-GORE, que implementa essas técnicas, é apresentada na sequência. A subseção seguinte provê maiores detalhes sobre técnicas de priorização de testes e por fim é apresentada a Matriz de Wieger que foi o método adotado para realizar o cálculo utilizado na priorização dos testes funcionais.

2.1 BDD - Behavior Driven Development

Como mencionado, o modelo de priorização aqui desenvolvido é pensado para ser usado no contexto de uso do BDD, ou Behavior Driven Development. O BDD foi criado a partir do Test Driven Development, por Dan North, para suprir algumas necessidades que o TDD não supria, como as que se referiam à especificações de software e documentação [23].

Logo, o BDD é uma técnica de elicitação de requisitos que, através dos testes de funcionalidade, acompanha o desenvolvimento do projeto criando o que se chama de *live documentation* e combinando os processos de validação e verificação de software [10].

O BDD é escrito em linguagem ubíqua para facilitar a leitura dos requisitos pelos *stakeholders* e assim entregar um produto de maior valor para o cliente. Isso foi pensado como forma de aumentar a interação entre os times de desenvolvimento e negócio. Assim, os requisitos do projeto são descritos como comportamentos nas chamadas histórias de usuário, que são escritas utilizando o formato "Dado, Quando, Então"(Given, When, Then).

Para verificar que as histórias de usuário foram implementadas, dentro do BDD cada uma delas possui testes de aceitação [10]. Um teste pode possuir vários cenários e ele só é considerado sucesso se todos os cenários forem executados sem erros. Para essa verificação, uma das ferramentas utilizadas é o Cucumber [7].

No Cucumber cada história de usuário é separada em um arquivo de funcionalidade (feature). Após a análise dos testes, a ferramenta Cucumber retorna os resultados para cada cenário: sucesso, falha ou pendência (estado onde o *step* ainda não foi definido).

2.2 GORE - Goal Oriented Requirements Engineering

Requirements Engineering ou Engenharia de Requisitos é uma subárea da Engenharia de Software, e tem como objetivo mediar a relação entre *stakeholders* e desenvolvedores de forma a estabelecer e manter os requisitos de um sistema [1].

Nesse contexto a abordagem GORE foi proposta para recuperar as intenções de negócio relacionadas aos requisitos do software [26]. Assim, busca-se utilizar o modelo de objetivos para ligar os requisitos aos objetivos (*goals*) dos *stakeholders* pois através dessa relação eles podem adquirir algumas vantagens.

Tais vantagens incluem: derivar os requisitos diretamente dos objetivos; utilizar os objetivos para recuperar a intenção humana e razão de existência por trás dos requisitos; adicionar uma estrutura mais compreensível aos requisitos através do modelo hierárquico de objetivos, entre outras [18].

O modelo hierárquico mencionado acima se dá pois dentro do modelo de objetivos é possível desdobrar os objetivos em outros

objetivos filhos ou em tarefas filhas. Tarefas seriam atividades a serem realizadas para garantir o funcionamento do sistema. Tais divisões são chamadas decomposições e cada abordagem possui suas próprias decomposições, sendo as mais comuns as decomposições AND e OR.

Uma decomposição AND indica que para o cumprimento do objetivo pai é necessário que todos os filhos participantes da decomposição sejam cumpridos. [11] A decomposição OR indica que apenas uma das opções precisa ser utilizada para cumprir o objetivo pai.

2.3 BDD-GORE

O BDD-GORE combina o BDD com a abordagem GORE [17]. Nela, *objetivos* são objetivos de negócio mais subjetivos do que os requisitos técnicos, e exatamente por isso carregam mais informações de negócio do que os requisitos recolhidos com o uso do BDD. Assim a abordagem GORE contextualiza melhor as tarefas de um sistema, ou seja, suas funcionalidades, por operacionalizar os objetivos de negócio da organização [15].

O BDD-GORE, por sua vez, faz uma relação entre as tarefas e as *features* de BDD. Cada tarefa é ligada à uma *feature*, que já tem seus testes escritos devido ao BDD e assim a ferramenta permite que as tarefas possuam estados. São esses: sucesso; falha; e pendente [17]. Esses estados são obtidos através da ferramenta Cucumber[7] que ao executar os testes, emite os resultados encontrados.

Para fazer a criação do modelo de objetivos, a abordagem utiliza a ferramenta piStar [25] com modificações para permitir a ligação das tarefas com as features [17]. Após feita essa ligação cada tarefa recebe uma cor relacionada a seu estado: vermelho para Falha, verde para Sucesso e azul para Pendente.

Através dessa relação das Tarefas com os testes de BDD, o modelo infere se o objetivo pai foi ou não cumprido. Essa abordagem permite um acompanhamento visual da evolução dos requisitos de um projeto e a manutenção de uma documentação viva que carrega informações de negócio além do que o BDD normalmente seria capaz.

Por esses pontos, esse estudo pretende utilizar a ferramenta BDD-GORE como base, adicionando a ela funcionalidades que permitam o gerenciamento da priorização dos testes de BDD, que já estão presentes no modelo de objetivos construído através do BDD-GORE.

2.4 Priorização de Testes Baseada Em Requisitos

Uma das técnicas de Priorização de testes (TCP) é a chamada Priorização de Testes Baseada em Requisitos, que utiliza os requisitos definidos pelo cliente para a ordenação dos testes do sistema [5]. Essa técnica ainda pode ser classificada por duas perspectivas, a chamada *Featured based testing prioritization* (ou *Functional Requirement Based TCP*) e a *Non Functional TCP*. Na primeira abordagem, utilizada aqui, busca-se ordenar os testes de forma a cobrir o maior número de requisitos funcionais, reduzindo o tempo de identificação de erros [21].

Nessa abordagem, utilizam-se propriedades relacionadas aos requisitos funcionais para fazer a priorização dos testes, como: a prioridade assinalada pelo cliente para o requisito; a complexidade indicada pelos desenvolvedores; a volatilidade dos requisitos; entre outras. Para o cálculo da prioridade final do teste, vários pesquisadores já sugeriram diversas técnicas que utilizam esses e outros fatores [21]. É comum que haja primeiro uma priorização dos requisitos para a partir daí se calcular a prioridade dos testes, como em PORT [14].

No desenvolvimento dessa ferramenta foi realizada uma pesquisa sobre os métodos existentes de priorização de requisitos para identificar o mais apropriado às necessidades do projeto. Tendo em vista a intenção de envolver explicitamente os *stakeholders* na priorização de testes e o número de requisitos normalmente presentes em projetos ágeis, decidiu-se por utilizar a Matriz de Wieger.

2.4.1 Matriz de Wieger. No levantamento de fatores, o método da Matriz de Wieger espera o envolvimento de pelo menos dois perfis: desenvolvedores e *stakeholders*. Os desenvolvedores são responsáveis por definir os fatores risco e custo, onde o primeiro está associado à possibilidade de não se cumprir o requisito e o segundo está relacionado à complexidade do requisito.

Por outro lado os *stakeholders* são responsáveis por definir o benefício e a penalidade de cada requisito. O benefício é a medida do quanto o requisito favorece o modelo de negócio, enquanto a penalidade indica o quão prejudicial seria se esse requisito faltasse ao negócio. Essa penalidade pode incluir tanto o lado negocial quanto técnico [31].

Antes de definir os fatores é necessário também definir os pesos de cada um. Por padrão, benefício e penalidade possuem o mesmo peso, mas isso pode ser modificado, o mesmo vale para risco e custo.

Uma vez que os pesos e os fatores estiverem definidos será calculado o Valor Total que é a soma do benefício e prioridades relativos. Assim, a Prioridade pode ser calculada como:

$$P = \frac{ValorTotal\%}{(custo\% * pesoC) + (risco\% * pesoR)} \quad (1)$$

Onde pesoC é o peso do custo e pesoR é o peso do risco.

Assim, o método da Matriz de Wieger tem os seguintes passos:

(1) Listar os requisitos a serem priorizados;
(2) Definir os valores estimados de benefício, penalidade dentro da escala 1-9. Essa definição é feita pelos stakeholders ;
(3) Definir os valores estimados de risco e custo dentro da escala 1-9. Essa definição é feita pelos desenvolvedores;
(4) Calcular os valores relativos, o Valor Total e a prioridade resultante;
(5) Ordenar a lista de requisitos de acordo com a prioridade.

Em relação ao número de requisitos suportados pelo método, ainda de acordo com Wieger [31], a técnica é voltada para projetos com maiores números de requisitos, o que é importante quando se fala de projetos ágeis. Como projetos ágeis são adaptados à mudanças e aceitam a inclusão e modificação de *features* [19] eles são projetos cujo número de requisitos pode crescer durante o período de desenvolvimento. Portanto, é importante que o método de priorização usado possa aceitar um maior número de requisitos.

Outro motivo para adotar a Matriz de Wieger é que a prioridade definida por ele é diretamente proporcional ao benefício e à penalidade relacionados ao negócio, sendo inversamente proporcional ao custo e ao risco de desenvolvimento. Como o objetivo da priorização de testes nessa extensão é diminuir o custo de desenvolvimento

e aumentar a satisfação do cliente com a entrega do MVP, o cálculo de Wieger foi considerado o mais adequado.

3 PROPOSTA DE SOLUÇÃO

Para realizar a priorização de testes nesse modelo decidiu-se criar uma extensão da ferramenta piStar, já modificada pela abordagem BDD-GORE. Como o BDD-GORE cria uma relação direta entre Tarefa, Requisito e Teste entende-se que para fazer a priorização dos testes é necessário fazer primeiro a priorização dos requisitos, onde a ordenação dos testes se dá como consequência.

Então, a implementação do cálculo da prioridade dos requisitos foi realizada utilizando a Matriz de Wieger, que resulta na prioridade das tarefas. Como cada Tarefa está relacionada a um teste funcional, esta se torna a prioridade do teste.

Além do cálculo de prioridade das Tarefas, essa ferramenta se propõe a sugerir uma sequência de desenvolvimento com base na prioridade dos testes. Para isso, considera-se as relações de decomposição AND e OR do modelo istar para fazer uma propagação da prioridade das tarefas. O modelo parte das folhas do grafo e propaga o valor de prioridade até chegar na raiz, definindo assim a prioridade dos Objetivos pais. A prioridade propagada para os objetivos será utilizada para fazer a sugestão de quais objetivos devem ser completados primeiro e quais são os testes que estão relacionados ao status desses objetivos. O cálculo é explicado mais à frente.

O projeto e a construção da solução são apresentados em detalhes nas próximas seções.

3.1 A Modelagem

O primeiro passo para estabelecer o modelo de prioridade foi decidir como seria feita a modelagem do modelo de objetivos. A premissa é que o projeto de desenvolvimento adota tanto o uso de BDD quanto o de Goal Model. O modelo de objetivos pode ser modelado na própria ferramenta graças às funcionalidades do piStar [25].

Nesse modelo considera-se para a criação do modelo de objetivos os mesmos conceitos de [8], focando principalmente na utilização de objetivos e tarefas. As principais entidades e dependências utilizadas nessa modelagem são:

- **Actor**: É a primeira entidade utilizada, sendo uma entidade ativa e autônoma que busca cumprir objetivos. Logo, *Goals* e *Tasks* não existem sem um *Actor* [8]. Um *Actor* pode ser o próprio sistema, por exemplo.
- **Goal**: É um objetivo de negócio ou um estado que o ator queira atingir. São definidos pelos *stakeholders* com base nos interesses de negócio. Um *Goal* pode ser desdobrado em outros *Goals* e *Tasks* através das dependências AND e OR.
- **Dependência AND**: Seguindo o exemplo de [17] se um *Goal* possui dependências AND entre seus filhos ele só é considerado concluído quando todos eles são atendidos.
- **Dependência OR**: Ainda seguindo [17] um *Goal* com uma dependência OR é considerado concluído ou tem o estado de sucesso, se pelo menos um de seus filhos também tem o estado de sucesso, não sendo necessário para a completude do *Goal* que os outros filhos atinjam esse estado.
- **Task**: São tarefas que representam os Requisitos do sistema. Espera-se para essa modelagem que cada *Task* possa ser ligada a uma única *feature* da ferramenta Cucumber. Assim cada *Task* é vista como um único requisito e pode receber o status gerado pelo Cucumber (Successo, Falha e Pendente).

Além das entidades, outro ponto importante da modelagem é a nomenclatura e a descrição que a acompanha. Elas definem a ordem de análise e em alguns casos até mesmo a relação de dependência entre entidades. Esses pontos são explicados nas próximas subseções.

3.1.1 Nomenclatura. No piStar a ordem de ligação do elemento filho, sejam eles Goal ou Task, com o pai é o que define a ordem em que eles seriam visitados ao considerar o diagrama como um grafo. Para evitar que isso se torne um problema, esse modelo utiliza uma nomenclatura específica para estabelecer a ordem que um elemento filho é avaliado em relação a um elemento pai. Essa nomenclatura é baseada nas mesmas regras de nomenclatura vistas em [20] e [12].

Assim, cada Goal com um prefixo G seguido por uma ID numérica única, dois pontos e uma descrição. Por exemplo: G1:Descrição.

As *Tasks* de cada *Goal* começam sempre com um prefixo T e uma ID numérica que sempre começa em 1 e aumenta conforme o número das *Tasks*. Na sequência é colocado dois pontos e a descrição da *Task*.

Como espera-se que as *Tasks* sejam equivalentes a um único requisito e a um teste, não se levou em consideração a nomenclatura para *Tasks* descendentes de outras *Tasks*.

3.1.2 Ordem de análise. A ordem de análise considera primeiro os elementos com ID's menores. Isso significa que em uma relação AND espera-se que o elemento de menor ID possua um status de sucesso antes dos elementos de maiores ID. Assim, os elementos de ID menores são considerados como uma dependência.

No entanto, na existência de uma decomposição AND onde a ordem dos filhos não é importante, a ideia é considerar primeiro os elementos de maior prioridade. Para identificar essa situação, utiliza-se na descrição do elemento uma notação de paralelismo, onde os prefixos dos elementos filhos são separados por . Exemplo: (G1#G2) significa que tanto G1 pode ser realizado primeiro quanto G2.

3.2 Cálculo da Prioridade

Uma vez que o modelo de objetivos estiver pronto, o próximo passo é ligar as tarefas com suas respectivas *features*, o que deve ser feito com a ajuda de um especialista. Para isso, adiciona-se à tarefa uma propriedade de nome "feature".

Quando essa propriedade é criada, os campos dos fatores utilizados por Wieger (benefit, penalty, cost e risk) também são criados e podem ser editados como pode ser visto na figura 1.

Para o cálculo da prioridade das tarefas é necessário adicionar o valor aos fatores de Wieger. De acordo com a matriz de Wieger [31] esses valores devem ser escolhidos numa escala de 1-9. Para os fatores *benefit* e *penalty*, como já mencionado, quem faz a escolha do valor é alguém ligado aos *stakeholders* enquanto que os fatores *cost* e *risk* são dados pela equipe de desenvolvimento.

Para esse cálculo a ferramenta possui uma função chamada *calculateTaskProperties*, cuja execução não é vista pelo usuário. Essa função recupera os valores dos campos mencionados acima e calcula para cada Tarefa: seu Valor Total, o custo relativo e seu risco

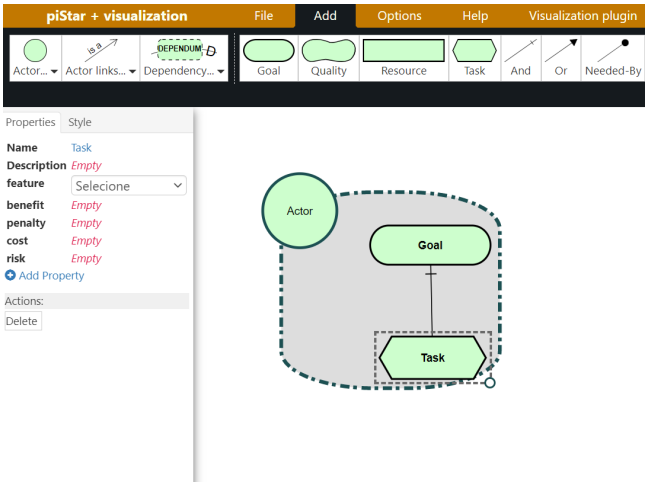

Figura 1: Imagem da ferramenta com a primeira propriedade de uma Task

relativo, como definido em [31]. Para isso ela também utiliza os pesos dos fatores, que como padrão são definidos como 1.

Esses pesos podem ser modificados pelo usuário no novo botão adicionado à ferramenta chamado Modificar Pesos, que pode ser encontrado no menu Help junto a outros botões do plugin BDD-GORE. Esse botão abre uma modal com diversos campos de texto para substituir os pesos padrão de cada fator.

Como a intenção inicial da modificação da BDD-GORE era adicionar uma função de prioridade, a função *calculateTaskProperties* foi adicionada no plugin do BDD-GORE sem a criação de um plugin novo, aproveitando-se de outras estruturas já criadas em [17].

Importante notar que essa função pode ser modificada, sendo possível utilizar outros métodos de priorização para fazer o cálculo.

Ao final do cálculo, todas as *Tasks* recebem a propriedade *priority* calculada através do método de Wieger e também recebem uma modificação visual explicada na próxima subseção.

3.3 Visualização de Prioridade

Após o cálculo da prioridade seu valor era adicionado ao elemento *Task* como propriedade, porém a consulta desse valor era ineficiente. Para poder consultar o valor da prioridade era necessário selecionar uma tarefa por vez, o que dificultava fazer uma comparação geral de prioridades. Por isso, sentiu-se a necessidade de implementar uma visualização para essa propriedade.

No desenvolvimento de software existe um campo chamado Visualização de Software, que promove pesquisas em relação aos efeitos da visualização aplicada no processo de desenvolvimento de software [2]. Vários estudos na área apontam benefícios e a importância de se utilizar a visualização no processo de Engenharia de Requisitos ou no próprio desenvolvimento de Software [6] [13] [16].

Sabendo disso, Pimentel et al. [24] desenvolveram para o próprio piStar um plugin de visualização de atributos. Os próprios autores citam a dificuldade de visualizar e analisar atributos de elementos de forma global no piStar como parte da motivação que levou ao desenvolvimento desse plugin de visualização. No plugin duas funcionalidades chamaram muito a atenção.

A principal funcionalidade permite que o usuário selecione um atributo para ser feita a visualização, que pode ser feita por cor, tamanho e utilização de ícones contendo o valor do atributo.

Para utilizar o plugin neste trabalho foi decidido adotar a classificação por utilização de ícones. Essa decisão se deve ao fato de que a base do BDD-GORE já utiliza um sistema de classificação por cores para indicar requisitos cujos testes tenham falhado ou constem como pendentes. Logo, utilizar duas classificações por cores deixaria a ferramenta confusa. A classificação por tamanho também foi desconsiderada pois atributos com valores muito pequenos tornam-se difíceis de serem visualizados, como menciona Pimentel et al. [24].

Foi realizada uma alteração neste plugin para atender ao contexto deste trabalho. Originalmente o plugin classifica visualmente todos os elementos que contenham o atributo escolhido. Como a intenção aqui era fazer a priorização dos testes (ou tarefas), decidiu-se evitar que os objetivos fossem classificados também, sendo necessário modificar o plugin. Assim, modificou-se a função de visualização para fazer apenas a classificação dos elementos do tipo Task, caso essa opção seja selecionada.

A segunda funcionalidade adotada desse plugin foi uma modal que define qual atributo é utilizado para a priorização e adiciona esse atributo a todos os elementos do grafo. Com ela espera-se diminuir o trabalho braçal do usuário no uso de outros atributos.

Assim, o menu Visualization Plugin de Pimentel et al. [24] foi adicionado ao corpo do piStar utilizado no BDD-GORE com os botões Manage Attributes e Visualize Range. O Manage Attributes é o botão utilizado para adicionar o atributo e Visualize Range é o botão que abre a modal para fazer a visualização do atributo de forma manual.

No entanto, para facilitar o uso da ferramenta, o botão Calcular Prioridade já faz a classificação visual automática com base no atributo *priority* gerado pelo cálculo de Wieger. A classificação visual, como mencionados, utiliza a visualização de ícones. Assim, após o cálculo da prioridade já é possível visualizar os valores para cada uma das Tarefas, permitindo um gerenciamento global das prioridades dos testes.

Além do cálculo das prioridades, a ferramenta também se propõe a fazer uma sugestão automática dos objetivos e tarefas mais prioritários, como visto a seguir.

3.4 Sugestão de Sequências de Desenvolvimento

A sugestão de sequência de desenvolvimento busca sugerir uma ordem de objetivos e tarefas a serem completados para aumentar a satisfação do cliente com a entrega do MVP. Essa sugestão, também chamada sugestão de caminhos, é feita com base nas prioridades calculadas para os testes e requisitos. Assim cada sequência é uma lista de objetivos e tarefas ordenadas.

O conjunto das sequências sugeridas é fornecido de forma ordenada em uma modal para o usuário, como pode ser visto parcialmente na figura 2. As sequências são numeradas da primeira (mais prioritária) à última. Os passos envolvidos na criação das sequências sugeridas são vistos à seguir em Propagação de Prioridade e Seleção dos Caminhos.

here

Figura 2: Captura de parte da modal de sequências

3.4.1 Propagação de Prioridade. Para a criação das sequências é feita primeiro a propagação da prioridade dos elementos filhos para os elementos pais. A propagação começa pelas folhas que são sempre um tarefa e é feita considerando as relações:

AND: Como todos os filhos devem ser cumpridos para a completude do pai, considera-se que todos contribuem igualmente para a prioridade do elemento pai. Assim a prioridade do elemento pai se dá pela média da soma das prioridades dos elementos filhos.

OR: A relação OR no BDD-GORE admite que para a completude do pai, basta que um dos filhos esteja completo. No contexto em que se busca a completude dos objetivos para a entrega de um mínimo produto viável, decidiu-se fazer a propagação do filho que possua a maior prioridade, pois ao cumpri-lo, não é necessário, no cenário do MVP, que se cumpram os outros filhos em um primeiro momento.

3.4.2 Seleção dos caminhos. Uma vez que todos os objetivos tenham recebido uma prioridade através da propagação, acontece a seleção dos caminhos. Ela leva em consideração quatro fatores: status do elemento; tipo de ligação; ordem estabelecida; e prioridade.

Ao considerar o **status do elemento**, o algoritmo verifica se esse elemento já foi completado com sucesso. Para essa verificação, aproveitou-se o algoritmo original do BDD-GORE que faz a propagação dos status das *Features* pelo *objetivo Model*, como mencionado na Seção 2.3. Se o elemento tiver sido completado, ele ainda será incluído no caminho, mas quando o resultado da sugestão for exibido, ele será escrito na cor verde.

A decisão de manter os elementos concluídos no caminho foi para manter a funcionalidade de documentação. Os elementos que não tiverem sido completados aparecem com a cor vermelha, indicando que ainda precisam de atenção.

O próximo fator verificado é o tipo de ligação que o elemento tem com os filhos. Caso seja uma ligação OR, o elemento sugerido na sequência desse pai é o elemento de maior prioridade. Caso seja uma ligação AND verifica-se primeiro a descrição do elemento pai para identificar se a ordem dos filhos deve ser a ordem de nomenclatura.

Se a ordem for a de nomenclatura, os elementos filhos são ordenados conforme os prefixos e então a verificação ocorre de forma recursiva em cada um. Senão, ordena-se os elementos filhos pelo de maior prioridade e continua a checagem.

O algoritmo explora os nós do grafo através da utilização de um algoritmo DFS recursivo que realiza as verificações aqui mencionadas. Ao final da análise ele abre uma modal com as sequências numeradas para que o usuário possa considerá-las. Em cada sequência, os elementos já concluídos se apresentam pela cor verde, enquanto os não concluídos são apresentados em vermelho. Quando a sequência possui elementos não concluídos ainda é possível baixar um arquivo em formato JSON contendo todos os cenários que falharam ou constam como pendentes, para futuras análises da equipe.

É possível executar esse algoritmo mais de uma vez, assim supõe-se que, ao final de uma iteração, quando outros *objetivos* ou *Tasks* tiverem mudado seu status para sucesso, o usuário possa utilizar novamente a funcionalidade e verificar a nova ordem sugerida pela ferramenta. Assim a ferramenta pode ser utilizada durante todo o processo de desenvolvimento.

4 AVALIAÇÃO

Para avaliar a utilidade da extensão BDD-GORE foram feitos dois tipos de testes: Avaliação da Extensão por Especialistas e Corretude da Prioridade. Seus respectivos planejamentos e resultados são descritos a seguir.

4.1 Avaliação por Especialistas

Essa seção apresenta o planejamento e os resultados da avaliação da ferramenta feita por especialistas.

4.1.1 Planejamento. Para avaliar o uso da extensão, foram realizados experimentos com dois desenvolvedores com experiência comercial de 2 a 4 anos no uso do BDD, pois a intenção é que a ferramenta seja utilizada por uma equipe de desenvolvimento. Alguns fatores de interesse na avaliação eram fatores de usabilidade, pois há a intenção de que futuramente a ferramenta possa ser utilizada no ambiente profissional, mas também havia interesse na corretude dos resultados da ferramenta, como nos caminhos de desenvolvimentos sugeridos e nas prioridades levantadas.

A forma escolhida para validar tanto o uso quanto a corretude dos resultados foi a opinião do usuário. Para isso foi criado um questionário e algumas de suas perguntas foram baseadas nas perguntas sugeridas pelo System Usability Scale [29], criado por John Brooke em 1986 e ainda hoje bastante utilizado para avaliar produtos e serviços.

Antes das perguntas relacionadas à ferramenta foram também adicionadas ao questionário perguntas de contexto que buscam prover informações sobre a familiaridade dos usuários com os conceitos envolvidos na ferramenta, como BDD e o *Goal Model*. São elas:

- **QC1:** Quanto Tempo de Experiência você tem com BDD?
- **QC2:** Você utiliza alguma técnica de priorização de testes ou requisitos? Se sim, qual?.
- **QC3:** Você conhece ou já utilizou o Goal Model?.

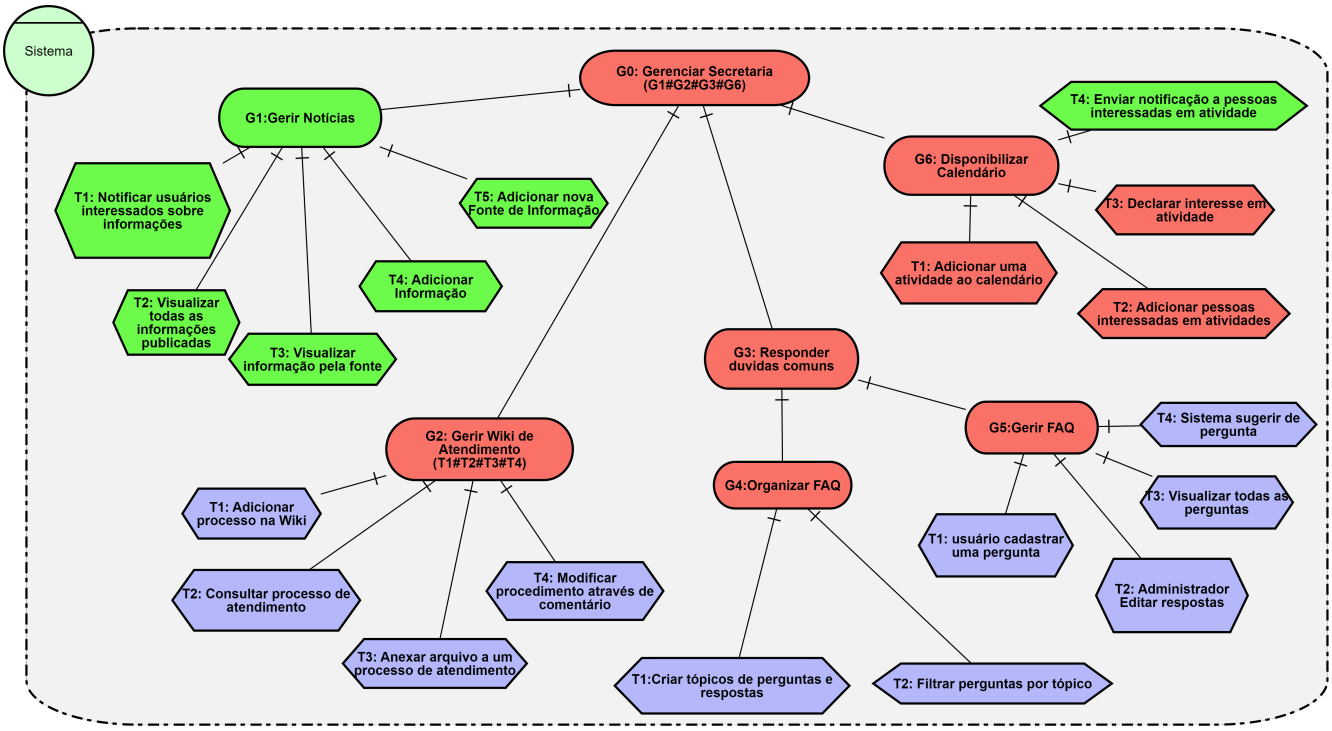

Figura 3: Prévia do modelo de objetivos utilizado para Testes

Para as outras seções, as perguntas devem ser respondidas em uma escala de 1 a 5, onde 1 significa Discordo Totalmente e 5 significa Concordo Totalmente. Para facilitar a interpretação dos resultados as perguntas foram desenvolvidas de forma que a resposta "Concordo Totalmente"na escala de 1 a 5 fosse a mais positiva para a pesquisa. Abaixo as onze perguntas que compõem o questionário, onde as primeiras sete questões são sobre usabilidade da ferramenta e as últimas quatro referem-se à sua funcionalidade:

- **Q1:** Achei a ferramenta fácil de usar.
- **Q2:** Utilizaria o sistema com frequência em meus projetos.
- **Q3:** Eu consegui utilizar o sistema sem ajuda técnica.
- **Q4:** A ferramenta envolve conceitos de fácil aprendizagem.
- **Q5:** O sistema é de simples utilização.
- **Q6:** Vejo possibilidade de utilização de ferramenta num ambiente profissional.
- **Q7:** O sistema tem utilidade ao longo do ciclo de vida do projeto.
- **Q8:** A visualização da prioridade das *Tasks* auxilia nas decisões de desenvolvimento.
- **Q9:** Seguindo um fluxo de TDD e considerando as prioridades calculadas eu concordo com a sugestão do sistema da sequência de atividades a serem desenvolvidas.
- **Q10:** As prioridades calculadas correspondem às expectativas.
- **Q11:** O sistema auxilia na documentação do projeto.

O questionário foi passado aos desenvolvedores após um teste prático da ferramenta onde foi fornecido a eles um modelo de objetivos já pronto, que pode ser visualizado na Figura 3. Esse *Goal Model* representa parte do sistema Secretaria PPGI que pode ser acessado em: https://github.com/EngSwCIC/secretaria_ppgi. Nele as tarefas já haviam sido relacionadas às *features* e com os valores benefit e penalty definidos. Assim pediu-se a cada desenvolvedor que definisse, para cada requisito, um valor de risk e cost e que fizessem a priorização dos testes e a simulação de caminhos de desenvolvimento.

As figuras 4, 5, 6, 7, 8, 9 trazem exemplos de BDD's do projeto refletidos no modelo, onde as figuras 4, 5 representam um teste com status sucesso, as figuras 6, 7 representam um teste com falha e 7, 8 representam um teste com *steps* pendentes.

4.1.2 Resultados. Nessa primeira sessão de testes, dois desenvolvedores de equipes diferentes avaliaram a ferramenta com base no mesmo projeto. De acordo com as perguntas de contextos, ambos tinham pelo menos 2 anos de experiência com BDD e já utilizavam técnicas de priorização de testes ou requisitos. Ambos já tinham conhecimento do modelo de objetivos, mas nunca o utilizaram.

Na Tabela 1 é possível ver as respostas de cada usuário para as perguntas aqui destacadas. Assim, nos primeiros dois testes a ferramenta parece promissora ao que se propõe, dada as respostas positivas referentes às perguntas de funcionalidade. Em relação à usabilidade, as notas parecem estar de acordo com os comentários finais deixados por cada usuário. O Usuário 1 estabeleceu nota 4 para a maioria dos fatores de usabilidade, exceto Q2 e Q3, que receberam notas mais baixas. Isso se reflete no comentário final que aponta a necessidade de grande inserção de dados e que isso pode

```
Feature: System administrator can register new source of information

    As a system administrator
    So that I can control sources of information
    I want to register a new source of information

Background: Start from the administrator page

    Given I am logged in as "administrator" on Secretaria_Ppgi
    When I follow "Informaçoes"
    Then I should be on "SecretariaPpgi informations page"
    When I press "Adicionar uma nova fonte de informação"
    Then I should be on "SecretariaPpgi Register Source of Inforamtion page"

Scenario: Register new source of information (happy path)

    When I fill the form with "information about the new source"
    And I press "Cadastrar fonte"
    Then I should recieve a message "Fonte cadastrada com sucesso"


Scenario: Add new source that has already been added (sad path)

    When I fill the form with "information about the new source"
    And I press "Cadastrar fonte"
    Then I should recieve a message "Não foi possível cadastrar fonte"
    And I should be on "SecretariaPpgi Register Source of Inforamtion page"
```

Figura 4: Teste da feature "Adicionar fonte de Informação"

```
Feature: System administrator can register new source of information
    As a system administrator
    So that I can control sources of information
    I want to register a new source of information

  Background: Start from the administrator page                          # features/Adm
    Given I am logged in as "administrator" on Secretaria_Ppgi           # features/ste
      administrator
    When I follow "Informaçoes"                                          # features/ste
      Informaçoes
    Then I should be on "SecretariaPpgi informations page"               # features/ste
      SecretariaPpgi informations page
    When I press "Adicionar uma nova fonte de informação"                # features/ste
      Adicionar uma nova fonte de informação
    Then I should be on "SecretariaPpgi Register Source of Inforamtion page" # features/ste
      SecretariaPpgi Register Source of Inforamtion page

  Scenario: Register new source of information (happy path)        # features/AdminControl.
    When I fill the form with "information about the new source"   # features/step_definitio
      information about the new source
    And I press "Cadastrar fonte"                                  # features/step_definitio
      Cadastrar fonte
    Then I should recieve a message "Fonte cadastrada com sucesso" # features/step_definitio
      Fonte cadastrada com sucesso

  Scenario: Add new source that has already been added (sad path)          # features/Admi
    When I fill the form with "information about the new source"           # features/step_
      information about the new source
    And I press "Cadastrar fonte"                                          # features/step_
      Cadastrar fonte
    Then I should recieve a message "Não foi possivel cadastrar fonte"     # features/step_
      Não foi possivel cadastrar fonte
    And I should be on "SecretariaPpgi Register Source of Inforamtion page" # features/step_
      SecretariaPpgi Register Source of Inforamtion page
```

Figura 5: Resultado do Cucumber para a feature "Adicionar fonte de Informação"

atrapalhar a utilização da ferramenta. Já as respostas do Usuário 2 em relação a usabilidade são mais otimistas, o que combina com seu comentário que aponta que a ferramenta pode ser de uso também para *stakeholders* não técnicos.

É interessante notar que para a pergunta Q8 ambos os usuários concordaram que a visualização das prioridades de teste auxilia na decisão de desenvolvimento, o que vai ao encontro dos trabalhos na literatura que impulsionaram a adição do *plugin* de visualização nesse trabalho.

Em relação à Q9, durante o uso da ferramenta ambos os usuários perceberam que se as dependências entre as tarefas no sistema fossem modeladas de forma errônea, os caminhos de desenvolvimento sugeridos poderiam ser afetados negativamente. É possível ver tanto na Figura 3 quanto na lista de requisitos mencionada na seção 4.2 que em G1, T4: Adicionar informação, vem depois de

```
#language: pt

Funcionalidade: Usuário pode adicionar pessoa interessada em uma atividade

    Como um administrador, para que eu possa informar a comunidade, eu gostaria de adicionar
    pessoas interessadas em cada atividade

    # Cenário Feliz
    Cenário: Adicionar pessoa em atividade
        Dado que estou logado com usuário "admin@admin.com" e senha "admin123"
        Dado que estou na página "Atividades"
        Quando eu clico em "Adicionar integrante"
        Então eu devo ser redirecionado para "Cadastrar_novo_integrante"
        Quando preencho o campo "Nome" com "Teste"
        E preencho o campo "Email" com "teste@teste"
        Então eu devo ser redirecionado para "Atividades"
        E eu devo ver "Pessoa adicionada à atividade com sucesso"

    #Cenário Triste
    Cenário: Página não encontrada
        Dado que estou logado com usuário "admin@admin.com" e senha "admin123"
        Dado que estou na página "Atividades"
        Quando eu clico em "Adicionar pessoa interessada"
        Então eu devo ser redirecionado para "404"
        E eu devo ver "Página não encontrada"
```

Figura 6: Teste a feature "Adicionar pessoa interessada em atividade"

Figura 7: Resultado do Cucumber para a feature "Adicionar pessoa interessada em atividade"

```
Funcionalidade: Anexar arquivo a um processo existente
    Como um secretário,
    para que eu possa detalhar mais os processos da Wiki,
    eu gostaria de anexar um ou mais documentos em um processo já existente

    Contexto:
        Dado que eu esteja conectado como usuario "alicinha@gatinha.com", "123456", "secretario"
        E que esteja na página "Wiki de Atendimento"
        E clico no botão "Editar"

    Cenário: Anexar um arquivo válido
        Quando envio o arquivo "document.xml"
        E clico no botão "Confirmar"
        Então sou redirecionado para a página "Processo"
        E recebo uma mensagem de sucesso

    Cenário: Anexar arquivo inválido
        Quando envio o arquivo "Pabllo_Vittar_seu_crime.mp4"
        E clico no botão "Confirmar"
        Então recebo uma mensagem de erro
```

Figura 8: Teste a feature "Anexar arquivo ao processo já existente"

G1:T3 e G1:T2 que estão relacionadas com visualizar informações. Como na descrição de G1 não há notação de paralelismo, infere-se que primeiro devem ser cumpridas as tarefas T2 e T3 para depois cumprir a tarefa T4. Por mais que um banco já possa estar populado de forma a exibir informações, ambos os desenvolvedores concordaram que a modelagem errônea pode levar a uma sugestão de caminhos menos intuitiva.

Similarmente Q10 também recebeu uma nota 4. Foi apontado pelos desenvolvedores que se a atribuição dos fatores for feita de

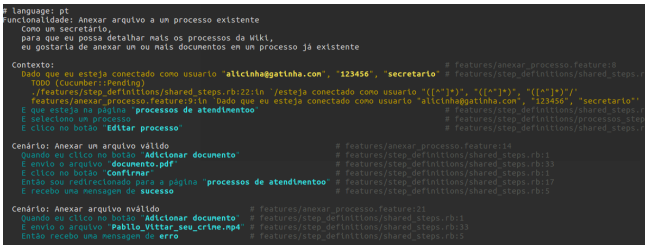

Figura 9: Resultado do Cucumber para a feature "Anexar arquivo ao processo já existente"

Tabela 1: Resultados Teste de Uso

Pergunta	Usuário 1	Usuário 2
Q1	4	5
Q2	3	5
Q3	2	4
Q4	4	5
Q5	4	5
Q6	4	5
Q7	5	5
Q8	5	5
Q9	4	4
Q10	4	4
Q11	5	5

forma incoerente, o resultado da prioridade calculada pela ferramenta também será incoerente com a realidade. No entanto, isso já era esperado desde a adoção do método de Wieger, onde o próprio autor admite [31] que é possível que os fatores sejam dados de forma a manipular ou induzir as prioridades resultantes a um erro, de forma que a aplicação desse método depende da experiência e integridade dos *stakeholders* e desenvolvedores ao distribuírem os fatores.

4.2 Corretude da Prioridade

Nesta seção está descrito o planejamento da avaliação da corretude da prioridade e faz-se uma discussão dos resultados.

4.2.1 Planejamento. Para avaliar a corretude da prioridade, além do questionário para o usuário, foram feitas algumas simulações utilizando uma tabela do Microsoft Excel contendo alguns requisitos e seus respectivos fatores de Wieger. Os resultados dessa tabela foram comparados com os fornecidos pela ferramenta. Como a ferramenta faz o arredondamento da prioridade, o resultado no Excel também foi arredondado com a função ROUND.

Na primeira simulação todos os fatores tinham peso 1. Na segunda, mantiveram-se os valores dados aos fatores na primeira simulação e o peso do fator Penalty e Risk foram modificados sendo, respectivamente, 4 e 3.

Os requisitos utilizados foram:

- **G1: T1** Notificar usuários interessados sobre informações
- **G1: T2** Visualizar todas as informações publicadas
- **G1: T3** Visualizar informação pela fonte
- **G1: T4** Adicionar Informação
- **G1: T5** Adicionar nova Fonte de Informação
- **G2: T1** Adicionar processo na Wiki
- **G2: T2** Consultar processo de atendimento
- **G2: T3** Anexar arquivo a um processo de atendimento
- **G2: T4** Modificar procedimento através de comentário

Novamente os requisitos são baseados em parte do projeto Secretaria PPGI. Na Tabela 2 é possível consultar os valores dos fatores para cada requisito nessa simulação.

Tabela 2: Fatores dos Requisitos para Simulação

Requisito	Benefit	Penalty	Risk	Cost
G1: T1	8	5	1	1
G1:T2	7	9	1	1
G1:T3	7	7	3	2
G1:T4	8	9	2	2
G1:T5	9	9	2	2
G2: T1	8	9	1	1
G2:T2	8	9	1	1
G2:T3	6	8	3	2
G2:T4	7	4	2	3

4.2.2 Resultados. Foram realizadas algumas simulações com o cálculo da prioridade de Wieger comparando os resultados dados pelo cálculo em uma planilha do Excel com o cálculo da ferramenta. É possível ver pelas Tabelas 3 e 4 que o resultado dado pela ferramenta em duas casas decimais é o mesmo resultado conseguido através do Excel para o mesmo número de casas. Assim, pode-se considerar confiável a parte matemática do cálculo de prioridade da ferramenta.

Tabela 3: Resultado de Prioridade: Primeira Simulação

Requisito	Prioridade Excel	Prioridade Ferramenta
G1: T1	0,73	0,73
G1:T2	0,9	0,9
G1:T3	0,32	0,32
G1:T4	0,48	0,48
G1:T5	0,5	0,5
G2: T1	0,95	0,95
G2:T2	1,01	1,01
G2:T3	0,32	0,32
G2:T4	0,25	0,25

No entanto, como mencionado anteriormente a coerência da prioridade vai depender dos valores fornecidos à mesma pelo usuário.

5 TRABALHOS RELACIONADOS

Como mencionado na Introdução desse artigo, a priorização e minimização da suíte de testes é uma área interessante por permitir minimizar o custo de desenvolvimento. Nesse contexto, a priorização de teste baseado em requisitos pode ser utilizada para melhorar

Tabela 4: Resultado de Prioridade: Segunda Simulação

Requisito	Prioridade Excel	Prioridade Ferramenta
G1: T1	0,32	0,32
G1:T2	0,49	0,49
G1:T3	0,15	0,15
G1:T4	0,25	0,25
G1:T5	0,26	0,26
G2: T1	0,5	0,5
G2:T2	1,51	1,51
G2:T3	0,16	0,16
G2:T4	0,12	0,12

a verificação da implementação de requisitos como mostrado por Butool et al. [27].

Em Butool et al. [27], os pesquisadores propõem um método de priorização de testes de regressão que calcula o peso do teste baseado na fração do requisito que ele cobre. Para isso, eles primeiramente fazem uma priorização dos requisitos e após calculados os pesos dos testes, eles assinalam a cada teste uma prioridade. Através do método proposto eles conseguem aumentar a detecção de falhas. Diferentemente de Butool et al. [27], o presente trabalho faz a priorização de testes de aceitação. Dessa forma cada requisito se mostra ligado a um único teste contendo vários cenários, de forma que aqui se considera que cada teste cobre por completo seu respectivo requisito.

Outro trabalho que também se utiliza da priorização através de requisitos é o de Reider et al. [19]. Nele, os pesquisadores consideram o contexto de desenvolvimento no método ágil e, para diminuir o custo de testagem, desenvolvem um método de geração e priorização de testes baseados em requisitos de comportamento. Para a priorização de testes, eles combinam a priorização de requisitos de PORT [14] com métricas usadas para verificar a cobertura de modelos (UML, Máquinas de estado, etc). Eles se utilizam do método PORT para fazer inicialmente a priorização do requisito considerando três dos fatores mencionados por Srikanth [14]: volatilidade dos requisitos, prioridade assinalada pelo cliente e complexidade dos requisitos. Após o cálculo da prioridade de cada requisito, as prioridades dos testes se baseiam em quantos requisitos cada teste cobre.

O método PORT chegou a ser considerado como método de priorização desse trabalho, no entanto como cada requisito do *Goal Model* está relacionado a um único teste, concluiu-se que a propagação da prioridade de requisito para testes feita por PORT não adicionava muito valor a essa extensão. Assim, aqui, diferentemente de em Reider et al. [19], considerou-se que a prioridade dada ao requisito seria, consequentemente, a prioridade recebida pelo teste.

Sendo assim, foi necessário uma pesquisa para identificar um método de priorização de requisitos. Entre os identificados, três chamaram mais atenção, sendo eles Volere, AHP e Wieger.

Volere [30] foi um dos primeiros métodos considerados por já ser conhecido no mercado. No entanto, no ponto de vista de Volere o uso de user stories pode gerar problemas, de forma que o autor cria seu próprio método de elicitação de requisitos. Como o contexto desse trabalho considerava a utilização de BDDs, concluiu-se que Volere não seria o melhor método para se utilizar.

Considerou-se então a utilização de AHP[32] ou Wieger [4]. No entanto, apesar de AHP apresentar resultados mais confiáveis, ela tem um problema de escalabilidade e funciona melhor para um pequeno número de requisitos devido à quantidade de comparações necessárias para o cálculo. Como esse trabalho visava a análise de pelo menos um número médio de requisitos e considerando os outros motivos mencionados durante esse trabalho, decidiu-se utilizar Wieger.

A priorização de requisitos utilizando Wieger com a ferramenta do piStar já havia sido realizada pelo menos uma vez. Em Flório et al.[9] os autores utilizam a ferramenta piStar como base para adicionar um modelo de priorização de requisitos. Para o teste, eles utilizam duas técnicas: alocação de cem dólares e a matriz de Wieger. Para o cálculo da prioridade final eles também consideram o peso de cada *stakeholder*. Esse trabalho usou Flório et al.[9] bastante como base, usando um processo similar à priorização de requisitos para a priorização dos testes. Porém a relação direta dos requisitos com os testes só foi possível devido à ligação fornecida pelo BDD-GORE entre o modelo de objetivos e os testes de BDD, que não existe na abordagem original de Flório et al.[9]. Essa ligação permitiu que o presente trabalho também apresentasse a sugestão de sequências de desenvolvimento.

Na seção seguinte, os resultados dos testes com a ferramenta são utilizados para analisar as contribuições das trazidas pela presente extensão.

6 CONCLUSÃO

Este artigo se propôs a trazer um modelo para auxiliar com o problema de priorização de testes no contexto ágil. Assim, decidiu-se que o propósito dessa priorização seria a criação de um MVP que diminuísse a insatisfação do cliente com possíveis entregas parciais. Para isso, a priorização dos testes foi baseada nos requisitos do sistema, combinando conceitos de BDD e da abordagem GORE. O resultado foi uma extensão da ferramenta BDD-GORE que adiciona à ela funções para o cálculo de prioridade. Essa extensão pode ser encontrada em: https://github.com/NayRSilva/bdd2Goal. Os testes realizados com a ferramenta demonstraram a corretude do cálculo proposto e o potencial de uso da ferramenta ao analisar as respostas dos usuários aos questionários.

No entanto, é importante considerar as ameaças à integridade. Muito desse modelo de priorização é dependente do uso e decisões humanas, pois o processo de desenvolvimento de software e gerenciamento de requisitos depende muito dessas decisões. Assim há algumas ameaças aos resultados da ferramenta dependendo do jeito que ela é utilizada, como visto até mesmo na seção 4.

O próprio cálculo da Matriz de Wieger se baseia em fatores subjetivos que podem ser manipulados pelos *stakeholders* e desenvolvedores para conseguir resultados específicos [31]. Além disso a utilização da escala de 1-9 para a decisão de fatores pode mudar a depender do usuário. A definição de custo ou complexidade também costuma depender do nível de conhecimento do desenvolvedor, então é possível que não haja uma precisão ou coerência se o mesmo projeto é utilizado por duas pessoas com entendimentos diferentes.

Adicionalmente, uma modelagem equivocada do sistema pode prejudicar a sugestão de caminhos. Isso foi exemplificado pelo modelo de objetivos utilizado na situação dos testes. Assim, caso um elemento seja declarado como dependente de outro em uma ordem incorreta, a sequência sugerida pela ferramenta pode prejudicar a lógica de desenvolvimento. Como o foco desse trabalho são os testes, e muitas dependências podem ser simuladas através de *mocks*, ainda é possível realizar a implementação de uma sequência contra intuitiva, no entanto isso não passou despercebido pelos usuários de teste.

Ainda assim, a avaliação dos times de desenvolvimento mostra que a ferramenta é promissora. Ambos os desenvolvedores concordaram que a ferramenta facilita a comunicação da prioridade de tarefas aos *stakeholders* e que as prioridades dada pela ferramenta auxiliam nas decisões de desenvolvimento.

Ainda se mostra necessária a realização de mais testes, principalmente para uma análise estatística da ferramenta. Adicionalmente, os testes de uso presentes nesse artigo utilizaram um projeto que já havia terminado seu ciclo de desenvolvimento e do qual os usuários não tinham conhecimento prévio.

Adicionalmente, por conta do tempo disponível dos participantes, apenas alguns módulos desse sistema foram utilizados para a geração do modelo de objetivos e do BDD. Logo, mostra-se interessante que se realize um teste da ferramenta em algum projeto de desenvolvimento ainda em curso. Dessa forma, é possível verificar o uso dessa extensão em tempo real por uma equipe que entenda melhor os objetivos de negócio do projeto.

Em relação a trabalhos futuros, há espaço para melhorias na ferramenta. Um dos desenvolvedores comentou que seria interessante ver a prioridade dos objetivos além da prioridade das tarefas. No entanto como esse trabalho se foca na prioridade de testes, o valor propagado que dá origem à prioridade dos objetivos não leva em conta fatores e métodos utilizados na priorização de objetivos de negócio. Assim, optou-se por ocultar esse valor pois ele não reflete a verdadeira priorização de objetivos negociais. Futuramente, é possível adicionar à ferramenta um novo módulo de cálculo de prioridades de objetivos que não dependa da extensão apresentada aqui. Outra ideia seria uma metodologia que combinasse ambas as abordagens. Além disso, uma melhoria interessante também seria a inclusão de um módulo que permitisse a modificação do cálculo de prioridade através da interface.

REFERÊNCIAS

[1] 2018. Systems and software engineering – Life cycle processes – Requirements engineering. *ISO/IEC/IEEE 29148:2018(E)* (2018), 1–104. https://doi.org/10.1109/IEEESTD.2018.8559686

[2] Zahra Shakeri Hossein Abad, Mohammad Noaeen, and Guenther Ruhe. 2016. Requirements Engineering Visualization: A Systematic Literature Review. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 6–15. https://doi.org/10.1109/RE.2016.61

[3] Khaled AbdElazim. 2020. A Framework for Requirements Prioritization Process in Agile Software Development. *Journal of Physics: Conference Series* (2020). https://doi.org/doi:10.1088/1742-6596/1454/1/012001

[4] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz'ri Mahrin. 2014. A systematic literature review of software requirements prioritization research. *Information and Software Technology* 56, 6 (2014), 568–585. https://doi.org/10.1016/j.infsof.2014.02.001

[5] Om Prakash Sangwan Anu Bajaj. 2019. A Systematic Literature Review of Test Case Prioritization Using Genetic Algorithms. *IEEE Access* (2019). https://doi.org/10.1109/ACCESS.2019.2938260

[6] John R Cooper, Seok-Won Lee, Robin A. Gandhi, and Orlena Gotel. 2009. Requirements Engineering Visualization: A Survey on the State-of-the-Art. In *2009 Fourth International Workshop on Requirements Engineering Visualization*. 46–55. https://doi.org/10.1109/REV.2009.4

[7] Cucumber 2022. Cucumber. https://cucumber.io/docs/cucumber.

[8] Fabiano Dalpiaz, Xavier Franch, and Jennifer Horkoff. 2016. iStar 2.0 Language Guide. (05 2016).

[9] Cinthya Flório, Maria Lencastre, João Pimentel, and João Araujo. 2019. *iStar-p: A Modelling Language for Requirements Prioritization*. 540–548. https://doi.org/10.1007/978-3-030-33223-5_44

[10] Armando Fox and David Patterson. 2021. *Engineering Software as a Service: An Agile Approach Using Cloud Computing* (2nd. ed.).

[11] Genaína N. Rodrigues Alessia Knauss João Paulo C. de Araújo Hugo Andrade Raian Ali Gabriel S. Rodrigues, Felipe P. Guimarães. 2019. GoalD: A Goal-Driven deployment framework for dynamic and heterogeneous computing environments. *Information and Software Technology* 111 (2019), 159–176. https://doi.org/10.1016/j.infsof.2019.04.003

[12] GODA 2022. GODA. https://github.com/lesunb/pistarGODA-MDP.

[13] Orlena C.Z. Gotel, Francis T. Marchese, and Stephen J. Morris. 2008. The Potential for Synergy between Information Visualization and Software Engineering Visualization. In *2008 12th International Conference Information Visualisation*. 547–552. https://doi.org/10.1109/IV.2008.56

[14] Hyunsook Do Hema Srikanth, Charitha Hettiarachchi. 2016. Requirements based test prioritization using risk factors: An industrial study. *Information and Software Technology* (2016). https://doi.org/10.1016/j.infsof.2015.09.002

[15] Z. Shen J. Lin, H. Yu and C. Miao. 2014. Using goal net to model user stories in agile software development. *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (2014). https://doi.org/10.1109/SNPD.2014.6888731

[16] Rainer Koschke. 2003. Software Visualization in Software Maintenance, Reverse Engineering, and Reengineering: A Research Survey. *Journal on Software Maintenance and Evolution* 15 (03 2003), 87–109. https://doi.org/10.1002/smr.270

[17] Fábio Barros Leal. 2019. *Uma abordagem usando features BDD e Modelo de Objetivos para o desenvolvimento ágil de software*. Universidade de Brasília.

[18] Jun Lin, Han Yu, Zhiqi Shen, and Chunyan Miao. 2014. Using goal net to model user stories in agile software development. In *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. 1–6. https://doi.org/10.1109/SNPD.2014.6888731

[19] Jan Krause Martin Reider, Stephan Magnus. 2018. Feature-based testing by using model synthesis, test generation and parameterizable test prioritization. *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops* (2018). https://doi.org/10.1109/ICSTW.2018.00041

[20] Danilo Filgueira Mendonça, Genaína Nunes Rodrigues, Raian Ali, Vander Alves, and Luciano Baresi. 2016. GODA: A goal-oriented requirements engineering framework for runtime dependability analysis. *Inf. Softw. Technol.* 80 (2016), 245–264.

[21] Imran Ghani Seung Ryul Jeong Muhammad Hasnain, Muhammad Fermi Pasha. 2021. Functional Requirement-Based Test Case Prioritization in Regression Testing: A Systematic Literature Review. *SN Computer Science* 2 (2021). https://doi.org/10.1007/s42979-021-00821-3

[22] Muhammad Muzammal. 2016. Test-Suite Prioritisation by Application Navigation Tree Mining. *2016 International Conference on Frontiers of Information Technology (FIT)* (2016). https://doi.org/10.1109/FIT.2016.045

[23] Dan North. 2013. INTRODUCING BDD. *Better Software Magazine* (2013). http://dannorth.net/introducing-bdd/

[24] João Pimentel, Maria Lencastre, and Luiza Freire. 2021. Visualization of the Values of Requirements Attributes with Goal Models. *Cadernos do IME - Série Informática* 45 (06 2021), 8–23. https://doi.org/10.12957/cadinf.2020.56850

[25] Pistar, UFPE 2022. Pistar Tool. https://www.cin.ufpe.br/ jhcp/pistar/.

[26] Fabiano Dalpiaz e Paolo Giorgini Raian Ali. 2010. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering* 15 (2010).

[27] Muddassar Sindhu Qamar uz Zaman Rimsha Butool, Aamer Nadeem. 2019. Improving Requirements Coverage in Test Case Prioritization for Regression Testing. *2019 22nd International Multitopic Conference (INMIC)* (2019). https://doi.org/10.1109/INMIC48123.2019.9022761

[28] Priyanka Paygude Snehal Chaudhary Sonali Idate Samridhi Sachdeva, Akshay Arya. 2018. Prioritizing User Requirements for Agile Software Development. *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)* (2018). https://doi.org/10.1109/ICACCT.2018.8529454

[29] SUS, System Usability Scale 2022. *SUS*. Retrieved April 25, 2022 from https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[30] Volere 2022. Volere Prioritisation Analysis. https://www.volere.org/prioritisation-analysis/.

[31] Karl E. Wiegers. 1999. First Things First: Prioritizing Requirements.

[32] Wendi Wirasta, Hira Laksmiwati Soemitro, and Bayu Hendradjaya. 2016. Utilization of AHP method in elicitation process for Goal Oriented implementation using KAOS modelling. In *2016 International Conference on Data and Software Engineering (ICoDSE)*. 1–6. https://doi.org/10.1109/ICODSE.2016.7936144