T-AADSP Test: Um módulo de gerenciamento de testes que utiliza Redes Bayesianas para priorização de casos de teste

Vitor Souza Silva Santana*
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N, Barbalho
Salvador, Bahia
vitorsouzassantana@gmail.com

Antonio Carlos Santos Souza†
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N, Barbalho
Salvador, bahia
antoniocarlos@ifba.edu.br

Resumo

O presente trabalho visa apresentar uma ferramenta que utiliza Redes Bayesianas para a priorização de casos de teste em projetos de *software*. Além da priorização, o estudo propõe o gerenciamento dos defeitos que surgirem durante os lançamentos do projeto. Consequentemente, apontar o status de cada caso de testes executado. Com a aplicação da ferramenta, espera-se reduzir o tempo gasto na escolha e execução de casos de teste de um projeto de *software*.

Palavras-chave

Redes Bayesianas, Priorização de Casos de Testes, Otimização em Engenharia de *Softwate*, Otimização em Teste de *Software*, AADSP

1. INTRODUÇÃO

O aumento da presença de *softwares* no cotidiano faz com que cada vez mais a interação entre usuários e dispositivos computacionais se expanda. A interação também faz com que sejam descobertas novas necessidades de sistemas [1]. A demanda por novos *softwares* faz com que o mercado de desenvolvimento de sistemas eleve o grau de competitividade. Além disso, é um dos diferenciais na hora de se destacar da concorrência seja através da entrega de produtos de maior grau de qualidade possível [2].

Diante desse mercado competitivo, as indústrias de *software* estão em busca de métodos e processos que façam com que a produtividade dos seus projetos aumentem [1]. Dentro de um projeto é importante que seja entregue o que o cliente pediu mas, tão importante quanto, é entregar um *software* confiável e com a menor quantidade de erros possível [3].

*Graduando em Análise e Desenvolvimento de Sistemas.
†Doutor em Ciências da Computacão e Professor do Curso de Análise e Desenvolvimento de Sistemas.

A verificação da qualidade de um *software* desenvolvido é medida através de testes. Existem variados tipos de testes que visam verificar a confiabilidade do sistema como um todo, não somente a parte funcional do produto [4]. Por exemplo, há testes que tem como objetivo verificar o quanto de estresse o sistema pode receber, ou o quanto de carga que ele suporta [4].

Só que testar um *software* é um processo custoso, demorado. As vezes os testes selecionados não conseguem cobrir totalmente a funcionalidade, resultando em erros que chegam ao cliente final. Esses erros podem causar consequências desastrosas, como prejuízos financeiros ou de reputação, podendo até chegar a danos a integridade física das pessoas [1].

Para tentar minimizar este problema, o presente trabalho propõe o *T-AADSP Test*. Este é, um sistema baseado na abordagem AADSP que permitirá o gerenciamento do módulo de teste dos projetos de *softwares* de uma empresa. A AADSP é uma abordagem para implantação de processo de *software* adaptativa criada pelo grupo de pesquisa do IFBA LABRASOFT [5].

O *T-AADSP Test* permitirá que partes dos requisitos sejam entregues em *sprints* com os respectivos desenvolvedores que atuaram nesse requisito. Uma *Sprint* delimita os requisitos que foram alterados e os casos de testes executados em um momento de tempo. O sistema terá um módulo para cadastro e gerenciamento de *bugs* encontrados no sistema e outro para o cadastro dos casos de testes. O sistema permitirá o cadastro de desenvolvedores e os seus níveis de maturidade, juntamente com uma nota atrelada ao seu desempenho com o passar das entregas de funcionalidades.

Para realizar a priorização dos casos de teste, será desenvolvido um algoritmo de Redes Bayesianas para calcular o risco de *bugs* de um módulo. Serão utilizados dados reais de projetos em andamento e concluídos, de diferentes portes e em diferentes fases de desenvolvimento, para o treinamento do algoritmo de Rede Bayesiana. É esperado que o resultado da predição tenha um alto índice de acerto nas escolhas dos casos de teste para. Assim, a sua aplicação em projetos de desenvolvimento reais ajudará a equipe de testes a reduzir o tempo gasto na escolha dos casos de testes mais prioritários. Isto é, os casos de testes que têm mais chances de capturarem erros, reduzindo, consequentemente, o custo da etapa de testes[6].

Este artigo está dividido em seções que discutirão sobre diferentes assuntos: a **seção 2** apresenta a fundamentação

teórica do trabalho; a **seção 3** apresenta os trabalhos relacionados com o sistema proposto , A **seção 4** mostra as etapas de desenvolvimento do projeto. Em seguinda, a **seção 5** apresenta os aspectos do desenvolvimento do sistema; a **seção 6** descreve como o projeto foi estudado; a **seção 7** discutirá os resultados obtidos. Finalmente, a **seção 8** apresenta a conclusão do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

A seção 2 apresenta todos os assuntos necessários para a realização deste trabalho. A seção 2.1 explanará sobre o conceito de teste de *software* e suas principais características. A seção 2.2 discute o contexto da aplicação do teste de software nas metodologias de desenvolvimento. A seção 2.3 trata dos modelos de processos de software MPS.BR, CMMI e AADSP, com enfoque no processo de qualidade de software. Na seção 2.4 apresenta o conceito de redes bayesianas, as suas principais características, algumas ferramentas, vantagens e desvantagens, exemplos de sua aplicação e um detalhamento da Rede Bayesiana proposta.

2.1 Teste de Software

O teste de *software* pode ser definido como uma investigação feita, geralmente por um testador, de um *software*, em fase de desenvolvimento [1]. No teste geralmente se busca a presença e o reporte dos defeitos e falhas encontrados na aplicação [6] [7], e a sua correção pelos respectivos programadores do sistema. O teste tem como finalidade a entrega de um produto de melhor qualidade, ou seja, com o menor índice de erros possíveis para o usuário final [6] [8] [9].

Nos projetos de software o teste é um processo que, dependendo da metodologia de desenvolvimento, é aplicado depois da fase de programação dos requisitos (nas metodologias mais rígidas, como a cascata). Ele também pode ser realizado desde o início do projeto do software, sendo avaliado a todo momento por todos os membros da equipe (nas metodololologias ágeis) [10]. Enquanto isso, a realização dos testes depois da implementacão traz problemas por causa dos inúmeros erros acumulados devido a falhas de testadores (ação humana é suscetível a erros) [8]. A aplicação do teste durante todo o projeto previne a ocorrência de muitos erros, já que a qualidade está sendo controlada a todo tempo [6].

As falhas em um software podem ter várias origens, sendo as principais as falhas por resolução de problemas reais (solução que resolve apenas uma parte do problema), falha de escrita da linguagem de programação, e por último perda de rastreabilidade de código (impacto de mudança nas outras funcionalidades) [8]. Como essas falhas podem levar a prejuízos financeiros ou até a integridade de pessoas, é extremamente necessário realizar testes em softwares [6].

É impossível testar um software por completo, pois não há como garantir que todas as combinações do sistema foram testadas [1]. Apesar de não ser possível provar que o software está 100% correto, pelo menos a aplicação dos testes fornece evidências de que o que foi desenvolvido está em conformidade com as funcionalidades especificadas.

O custo de tempo e de dinheiro que a atividade de teste consome do projeto são grandes, chegando a quase 40% do gasto de todo o projeto. Isso ocorre devido a fase de execução dos testes e a maturação do sistema decorrente da correção de defeitos [3]. O custo da correção de um defeito tende a subir o quão mais tarde ele é encontrado. O defeito tem o custo aumentado em uma proporção de dez para cada fase em que ele não for corrigido [11]. Um defeito encontrado na criação do *design* do sistema e no requisito custa 10 vezes menos do que encontrado durate a codificação e 1000 menos do que encontrado no cliente [8].

As atividades de teste estão divididas em duas técnicas, quatro tipos e cinco níveis [1]. As técnicas de teste têm como objetivo aferir o funcionamento do sistema internamente ou externamente [11]. Já os tipos de teste têm a finalidade de testar o software através de um objetivo particular [12]. Os níveis dividem o teste de um software em camadas [13].

2.1.1 Definição de Erro, Defeito e Falha

Para evitar ambiguidades durante a comunicação no dia-a-dia durante o processo de teste ou em materiais lançados sobre teste de software, os conceitos de erro, defeito e falha foram separados e definidos, com base no ISTQB (*International Software Testing Qualification Board*) [8] [9] [4]. Na hierarquia dos termos, o erro produz um defeito, que por sua vez resulta em falhas no software [9]. A imagem 1 exibe visualmente a relação entre esses conceitos.

O erro é causado por alguma ação humana que produz um resultado incorreto. O erro é causado, por exemplo, por uma falha de programação em algum trecho de código ou por uma falha na análise de especificação de requisitos [8] [14].

O defeito por sua vez, é considerado o resultado de um erro, e impede que alguma parte do sistema não desempenhe a sua função corretamente. Um erro pode gerar vários defeitos em um código, ou vários erros podem vir a causar somente um defeito [8].

Figura 1: Representação da relação entre defeito, erro e falha [15].

Já a falha é caractetizada pelo comportamento operacional diferente daquele esperado pelo usuário [8]. Falhas também podem ocorrer por condições ambientais (salinidade, magnetismo, poluição). Elas podem, por exemplo, causar falhas em um sistema embarcado ou mudar a execução de um *software* devido ao tipo de exposição do *hardware* [9].

2.1.2 Técnicas de Testes

Como mencionado anteriormente, existem duas técnicas de execução de testes: testes de caixa preta e caixa branca [4]. Como mostra a figura 2, as duas tem o objetivo de avaliar o funcionamento do código de diferentes ponto de vista.

O teste de caixa preta consiste em testar o software sem ter nenhum contato com o funcionamento dos códigos que existem dentro do sistema. O testador apenas interage com a parte gráfica do sistema, dando entradas em dados e verificando as saídas das informações geradas [8]. O teste de

caixa preta tem como objetivo realizar as verificações dos requisitos funcionais do software.

Enquanto a técnica caixa preta realiza testes baseados na especificação, na técnica de caixa branca os testes são feitos diretamente no código-fonte do sistema. Para realizar esse tipo de teste, deve-se conhecer a estrutura interna do que será testado, para saber se o código escrito atende aos que foi pedido no sistema [9] [8] [4].

2.1.3 Tipos de Testes

Existem quatro tipos de testes que podem ser aplicados em um software: Funcionais, Não-Funcionais, Estruturais e de Manutenção [1]. Cada tipo de teste tem como objetivo realizar o teste de partes distintas de um projeto [4].

O Teste Funcional consiste em verificar se o sistema está funcionando de acordo com o que se espera dele, ou seja, se o que foi especificado realmente está sendo feito pelo programa. O teste funcional é feito através de ações externas ao sistema [8].

Os testes funcionais são, basicamente, a aplicação da técnica de caixa preta orientados por roteiros de testes que indicam as ações que devem ser feitas, os dados que devem ser inseridos e a resposta que o sistema deve retornar a essa ação [8].

Os Testes Não-Funcionais são realizados em um software para se obter a resposta desse software em variadas situações [3]. Como o próprio nome sugere, esse tipo de teste não tem o objetivo de verificar a funcionalidade do software e sim se ele consegue ser executado em situações extremas. Dentro dos testes não-funcionais podem ser aplicados testes de desempenho, carga, stress, segurança e de portabilidade [8].

O Teste Estrutural verifica como o sistema foi construído, através da análise dos código-fonte do sistema. Esse teste tem como objetivo averiguar se o código escrito está bem-documentado e se foram aplicadas boas práticas [1].

O Teste de Manutenção pode ser divididos em dois subtipos, teste de confirmação e teste de regressão: o teste de confirmação testa se algum problema foi encontrado e reportado para a equipe de desenvolvimento, e se esse problema foi corrigido [1]. Já o teste de regressão consiste em aplicar, em cada versão do sistema, todos os testes feitos anteriormente. Esse tipo de teste é importante para verificar se alguma nova mudança em uma parte do software não causou nenhum problema no restante do sistema [8] [4].

2.1.4 Níveis de Testes

Os testes estão divididos em cinco níveis [1], e cada um destes níveis tem a finalidade de testar o software em um determinado ponto de desenvolvimento, desde o início da escrita de códigos até a implantação no cliente [13]. Os níveis de testes podem ser unitários, de integração, de validação, de sistema e de aceitação [8] [17].

No teste unitário, o objetivo é verificar as menores unidades do software produzido. Nesse teste, procura-se no código erros de lógica e de implementação em cada unidade separadamente [17]. Geralmente, o próprio desenvolvedor do módulo que contém as unidades realiza os testes [8].

No teste de integração, deve-se testar a interação entre os diferentes módulos do sistema [4]. O teste de integração visa buscar consequentes erros associados a interação de dois módulos [17]. Por exemplo: Se o módulo A está integrado com o módulo B, no teste de integração deve ser testada a comunicação entre os módulos, e não a funcionalidade dos dois [9].

O teste de validação tem como objetivo realizar testes no software que está pronto para ser entregue para o cliente. O teste é feito pelos administradores do ambiente do cliente, e tem como objetivo verificar que a entrada em produção terá sucesso [1].

O teste de sistema analisa se o software que foi produzido atende todos os requisitos, funcionais ou não, necessários para a execução no lado do cliente [4]. Nessa etapa, os testadores fazem baterias de testes caixa-preta para verificar o comportamento do sistema a cada caso de uso final [8].

Já o teste de aceitação verifica se o software final está pronto para ser entregue ao cliente, e se todas as necessidades do cliente são atendidas [4]. Esse teste é feito juntamente com o cliente no ambiente em que o software será usado, e com o uso de dados especificados ou reais [1].

2.2 Otimização em Engenharia de software

A área de Engenharia de Software teve uma grande contribuição para o gerenciamento de desenvolvimento de softwares, promovendo várias mudanças importantes na estrutura de processo de software. A partir desses estudos, foram geradas normas e metodologias que guiaram todas as fases do desenvolvimento de um sistema [18].

Mas existem problemas que a área de Engenharia de Software tradicional não consegue resolver, ou não consegue resolver de forma satisfatória. Esses problemas geralmente são muito complexos, contém um alto número de possibilidades a serem escolhidas e a complexidade da resolução é altíssima [19]. Esses problemas podem ser resolvidos quando soluções matemáticas são aplicadas. As técnicas matemáticas utilizadas visam automatizar as formas de resoluções para obter a forma mais eficiente de se resolver um problema, buscando a melhor solução em um espaço de possíveis soluções [19].

Com a união da área de otimização matemática com a área de engenahria de software, surgiu uma nova área de pesquisa, denominada de Otimização em Engenharia de software ou, em inglês, *Search-based Software Engineering* (SBSE) [19] [18].

Dentro da área de Otimização em Engenharia de Software, existem várias subáreas que já estão sendo estudadas. Entre essas áreas, existe uma dedicada especialmente para a área de teste de software, denominada Otimização em Teste de Software [19] [18].

2.2.1 Otimização em Teste de Software

A subárea de Otimização em Teste de Software (em inglês, *Search-based Software Testing*) abriga todos os trabalhos feitos que tem como objetivo otimizar algum processo da etapa de Teste de Software. Conforme a Tabela 1, existem cinco principais processos: Geração de Dados de Teste, Seleção de Dados de Teste, Seleção dos Casos de Teste, Priorização dos Casos de Teste, Testes Não-Funcionais e Testes Funcionais [19].

Dentro da área de otimização em Teste de Software, há uma seção específica para trabalhos que tem como objetivo realizar a priorização de casos de testes de uma funcionalidade. Essa priorização tem como objetivo ordenar os casos de teste de forma que os mais significativos sejam executados primeiro. Isso é bastante importante quando não existe tempo ou recurso o suficiente para executar todos os casos de testes possíveis, sendo garantida a maior cobertura pos-

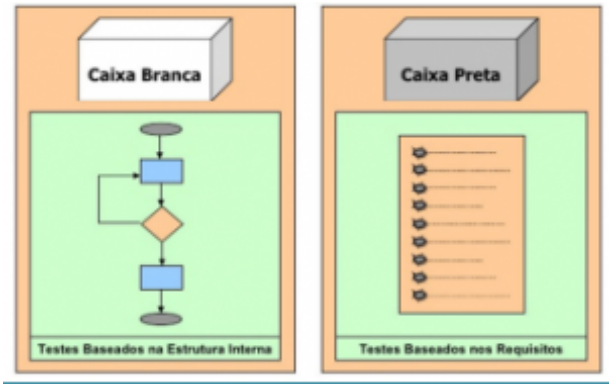

Figura 2: Representação visual da estrutura de um teste de caixa-branca e de caixa-preta [16].

Área da Engenharia de *Software*	Principais Problemas
Engenharia de Requisitos	Análise de Requisitos Seleção de Requisitos Planejamento de Requisitos
Teste de *Software*	Geração de Dados de Teste Seleção de Casos de Teste Priorização de Casos de Teste Testes Não Funcionais
Estimativa de *Software*	Estimativa de Tamanho Estimativa de Custo
Planejamento de Projeto	Alocação de Recursos Alocação de Pessoal
Otimização de Código-Fonte	Paralelização Otimização para Compilação
Manutenção de *Software*	Re-engenharia de *Software* Automated Maintenance
Otimização de Compilador	Alocação em Heap Tamanho de Código
Projeto de *Software*	Modularização

Tabela 1: Áreas já pesquisadas da área de otimização de Engenharia de Software [19]

sível de testes no sistema mesmo com a sua parada [19] [18]. Na tabela 2 podem ser vistas todas as áreas que tiveram algoritmos implementados para a otimização.

É importante informar que, tanto a subárea de Otimização em Teste de Software, quando sua área pai, a Otimização de Engenharia de Software, foram desenvolvidas para complementar as técnicas já estabelecidas na área da Engenharia de Software tradicional. As áreas têm como objetivo apenas resolver problemas que os métodos convencionais da

Problema de Teste de *Software*	Observações
Geração de Dados de Teste	Uso de algoritmos genéticos Formalização do problema Maximização dos caminhos Quantidade de vezes do teste Uso de tempera simulada
Seleção de Casos de Teste	Análise de cinco técnicas Abordagem multiobjetiva inicial Nova formulação multiobjetiva Seleção segura de casos de teste
Priorização de Casos de Teste	Relaçãode priorização e seleção Formalização do problema Tempo de execução como fator Comparação de técnicas Uso do GRASP Reativo
Testes Não Funcionais	Teste de performance Teste de tempo de execução Testes em contextos reais
Testes Funcionais	Teste de interação

Tabela 2: Problemas atacados na área de Otimizacao em Teste de Software [19]

Engenharia de Software tradicional não conseguiam, ou conseguiam, mas de forma insatisfatória [19].

2.3 Inteligência Artificial e Redes Baysianas

Uma das técnicas de IA que utilizam probabilidades são as redes bayesianas. Redes Bayesianas podem ser definidas como modelos que tentam explicar os motivos para a ocorrência de um evento ou prever suas consequências através da análise das variáveis envolvidas [20]. A técnica surgiu a partir de situações onde há um número grande de variáveis e se deseja saber a influência de uma variável não-direta entre as outras [21].

O renomado reverendo inglês Thomas Bayes (1701-1776) foi o idealizador do chamado teorema bayesiano, teorema esse que é a base das Redes Bayesianas [22]. Bayes definiu na sua equação que a probabilidade de um evento acontecer pode ser dada pela associação da probabilidade a priori junto com a probabilidade condicional, retornando, depois dos cálculos, a probabilidade do evento acontecer ou não (chamado de probabilidade a posteriori) [22].

Dentro da área da Inteligência Artificial, as Redes Bayesianas estão incluídas no que se pode chamar de subárea "Inteligência Artificial Probabilística", juntamente com a técnica de Redes Neurais e de Regressão Logística [21].

Uma das grandes vantagens das Redes Bayesianas é o fato dela permitir a visualização gráfica de relações, tanto causais quanto probabilísticas, das variáveis de um domínio [23]. Devido a essa característica, as Redes Bayesianas conseguem unir conceitos de teoria dos grafos para a visualização gráfica, teoria de probabilidades, de ciência da computação e de estatística [21].

2.3.1 *Teoria de Bayes*

Para ficar mais claro, eis um exemplo do teorema aplicado na área médica [24]: A meningite causa torcicolo em 50% dos casos, mas também se sabe que um caso de meningite atinge 1 em cada 50000 pessoas e a torcicolo, 1 em cada 20 pessoas. Considere P(M|T) como a probabilidade incondicional do paciente ter torcicolo/meningite respectivamente. Assim:

- **P(T e M) = 0,5**
- **P(M) = 1/50000**
- **P(T) = 1/20**

Aplicando a regra:

- **P(M|T) = (P(T|M)P(M))/P(T) =**
- **(0.5 x 1/50000)/(1/20) = 0.0002**

Então, a probabilidade de um paciente ter meningite continua pequena, mesmo com o torcicolo.

2.3.2 *Funcionamento*

Uma Rede Bayesiana é constituída de representações gráficas dos relacionamentos entre variáveis, [21] cuja representação pode ser vista na figura 4, e cada relacionamento representa uma dependência de causa entre as variáveis conectadas [25]. A rede se utiliza de grafos acíclicos direcionados (DAG) para a representação: um nó simples significa uma variável na rede e um arco identifica a relação de dependência entre duas variáveis conectadas. A figura **??** mostra as representações gráficas de um nó e de um arco.

Nas estruturas que formam a rede, deve existir somente um caminho de ligação de um nó a outro (nó pai / nó filho). Só é permitida a comunicação a dois nós conectados através de outros nós compartilhados com ambos. [21] [24] [25].

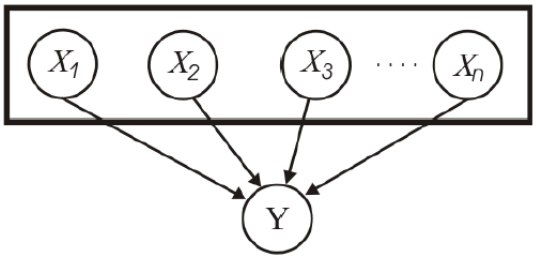

Figura 3: Exemplo de uma Rede Bayesiana [25]

Uma vez definida a rede, é necessário definir a tabela de probabilidades para cada nó [24]. Nessa tabela, fica registrada a probabilidade condicional do nó filho ligada diretamente aos valores inseridos no(s) nó(s) pai(s) [24] [21].

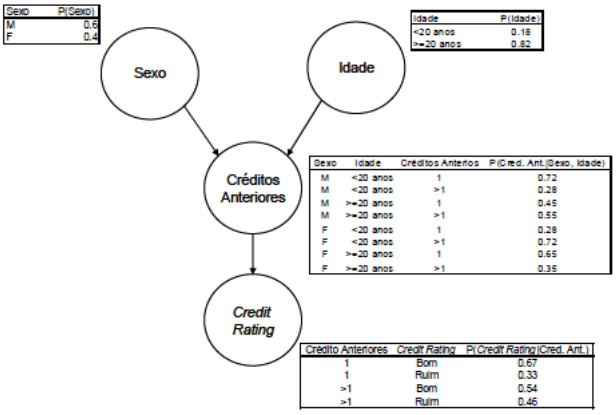

Figura 4: Exemplo de Rede Bayesiana de *Credit Score*, com as condicionais já definidas dos nós filhos (Créditos Anteriores e Credit Rating) [25]

2.3.3 *Uso das Redes Bayesianas*

Existem vários exemplos de uso de Redes Bayesianas nas áreas do conhecimento humano, principalmente na área de diagnóstico [24]. Eis alguns exemplos de uso:

- **Sistema BayesAR:** O sistema BayesAR foi um sistema desenvolvido que foi entregue como trabalho final de um doutorando no ano de 2006. O sistema consiste em uma Rede Bayesiana desenvolvida com o objetivo de reconhecer e classificar objetos de sistemas de Realidade Aumentada [25]. Para isso, ele utilizou três características: cor, forma e textura [25].
- ***Credit Score*:** Existem diversas empresas de crédito que utilizam algoritmos feitos a partir de Redes Bayesianas para estimar se determinada pessoa será boa pagadora ou não, por exemplo [21]. A *BayesCredit* é um sistema criado pela empresa Dinamarquesa Nykredit, especializada em financiamento de imóveis [21].
- **Predição de Reprovação de Alunos da modalidade EAD:** Foi apresentado no III Congresso Brasi-

leiro de Informática na Educação (CBIE 2014) um algoritmo de Redes Bayesianas que foi programado para fazer a predição de reprovação dos alunos matriculados nos cursos EAD [26]. Alguns dos atributos usados na rede foram: Situação acadêmica, Interações por Semana e o Fator de Empenho.

2.3.4 Rede Bayesiana proposta

A Rede Bayesiana proposta nesse trabalho e implementada dentro do *T-AADSP Tes*t foi primeiramente modelada no *Netica*. O *Netica* é um software feito pela *Norsys Software* que permite a criação de Redes Bayesianas através de interface gráfica. A representação gráfica da Rede Bayesiana pode ser vista na figura 5.

De certo modo, pode-se dizer que a Rede Bayesiana principal tem outras duas redes embutidas, onde uma irá predizer a hierarquia gerencial do requisito e a outra irá predizer a chance de um requisito ter *bugs*. Juntas, essas redes priorizam o requisito decidindo se, mesmo com uma alta chance de *bugs*, a sua execução é importante dentro da hierarquia dos projetos de software. As evidências da tabela de probabilidade de cada nó estará disponível futuramente no apêncide deste trabalho.

Os nós da Rede que são responsáveis por calcular a chance de *bugs* em um requisito são:

- **AlteraçõesRequisito:** Será fornecido quando o usuário vincular um requisito a *Sprint* que está em andamento. O usuário informará qual foi o impacto que as novas funcionalidades e/ou as correções de *bugs* causaram no requisito.

  A alteração pode ter até três níveis: quando o impacto das alterações não afetaram muito o que já tinha sido programado antes, diz-se então que as mudanças foram de Baixa Propoporção; quando as alterações impactaram as alguma das funcionalidades anteriores do requisito, diz-se então que a proporção é média. Quando houve um impacto imenso na maioria das funcionalidades ou em todas as funcionalidades, define a proporção como sendo alta.

- **MediaNivelDesenvolvedor:** A entrada desse nó será feita a partir da nota dos desenvolvedores que atuaram no requisito. Caso somente um desenvolvedor tenha atuado no requisito, somente a nota dele será utilizada; caso mais de um desenvolvedor tenha atuado, a média das notas dos desenvolvedores será dada como entrada.

  Se a nota resultante for menor que 4.9, a entrada no nó indicará que os desenvolvedores são pouco confiaveis. Se a nota resultante ficar entre 5.0 e 7.9, a entrada indicará que os desenvolvedores são confiaveis. Caso a nota seja maior que 8, os desenvolvedores serão considerados muito confiáveis.

- **ChanceDeImpactoNoRequisito:** A probabilidade desse nó é dada através dos nós pais **AlteraçõesRequisito** e **MediaNivelDesenvolvedor**. Ele indicará se a chance de impacto no requisito é alta, média ou baixa a partir dos resultados dos seus nós pais.

  Se, por exemplo, a alteração foi de alta proporção e o desenvolvedor é muito confiável, diz-se que a chance de impacto nas funcionalidades do requisito é alta, mas não tão alta devido a qualificação do desenvolvedor. Mas se a nota do desenvolvedor por menor, a criticidade de *bugs* no requisito será maior e por consequência, sua chance vai ser mais alta.

- **PorctCoberExecTestesRelAnt:** Esse nó indicará a o nível de cobertura de testes que uma *sprint* anterior teve. O cálculo da porcentagem se dará através do número de todos os casos de testes executados com sucesso pelo número total de casos de testes existentes daquele requisito.

  Se a porcentagem da cobertura for menor que 40%, a cobertura será considerada mínima; se a porcentagem estiver entre 40% e 69%, a cobertura será definida como sendo média. Acima de 70, a cobertura será considerada alta.

- **RiscoDeBugsNoRequisito:** A probabilidade desse nó é dado através dos nós ChanceDeImpactoNoRequisito e PorctCoberExecTestesRelAnt, e definirá a probabilidade de encontrar *bugs* no requisito.

  Caso a chance de impacto no requisito seja alta, mas a cobertura de testes da *sprint* anterior for alta também, a chance de encontrar *bugs* no requisito reduz consideravelmente. Caso a cobertura não tenha sido tão alta, pode ser então que *bugs* da *sprint* anterior não tenham sido detectados devido a baixa porcentagem e sejam detectados nessa *sprint*.

Já os nós da Rede que são responsáveis por calcular a hierarquia do requisito são:

- **NivelImporStakeholderRequisito:** Do mesmo jeito que um projeto tem um *stakeholder* vinculado, um requisito também tem um *stakeholder* vinculado. Um requisito tem prioridade sobre outro se a nota de prioridade do *stakeholder* vinculado também for maior.

  Se a nota do *stakeholder* do requisito estiver entre 1 e 5, a importância do requisito é baixa; se estiver entre 5.1 e 7.9, a a importância é considerada média. Acima disso, a importância é alta.

- **NotaPrioridadeRequisito:** Esse nó é representado pela nota do requisito. Define o quão importante um requisito é para um projeto de software.

  Se a nota de prioridade for menos que 4, a prioridade é definida como baixa; se a prioridade estiver entre 4 e 6,9, a prioridade é definida como média. Igual ou acima de 7, a prioridade é alta.

- **ImportanciaRequisito:** O nó é resultado dos nós NivelImporStakeholderRequisito e NotaPrioridadeRequisito. Ele indica o quão importante é o requisito frente a outros requisitos.

- **CustoProjeto:** O custo do projeto é fornecido pelo usuário quando um projeto é cadastrado. A depender da empresa, a mesma pode ter vários projetos de pequeno, médio, e grande porte, e o custo desses projetos também são diferentes entre si.

  Quando um projeto tem um custo inferior a 30 mil, a entrada que vai ser dada no nó é de que o projeto tem baixo custo. Se o valor do projeto estiver entre 30 e 100, o projeto é considerado de médio custo, e se o valor for acima de 100 mil, o custo do projeto é alto.

- **DeadlineDoProjeto:** A *Deadline* (prazo de entrega) do projeto indica quando o projeto será entregue ao cliente. Se o prazo de entrega do projeto estiver com até 3 meses depois da data atual, o projeto é classificado como tendo *deadline* curto. Se a data de entrega estiver entre 3 e 6 meses, é verificado que o prazo não está tão crítico e o *deadline* é classificado como médio. Caso o prazo esteja acima de 6 meses da data atual, o projeto é classificado como tendo *deadline* longo.

- **NivelImportStakeholderProjeto:** Dentro do sistema, cada *stakeholder* tem uma nota de prioridade vinculado a ele. Um *stakeholder* pode estar vinculado a um ou mais projetos dentro do sistema. Um projeto tem prioridade sobre o outro se a nota de prioridade do *stakeholder* vinculado a ele for maior.

  Nesse nó, o projeto a qual o requisito pertence será dado como entrada; se o *stakeholder* desse projeto tiver uma nota entre 1 e 5, a importância do projeto é baixa; se estiver entre 5.1 e 7.9, a a importância é considerada média. Acima disso, a importância é alta.

- **ImportanciaProjeto:** O nó é resultado dos três nós DeadlineDoProjeto, NivelImportStakeholderProjeto e CustoProjeto. Ele determina a hierarquia do projeto frente a outros projetos.

- **StatusRequisito:** A probabilidade desse nó é dado pelo resultados dos nós ImportanciaRequisito e ImportanciaProjeto. Esse nó indica qual irá ser o a prioridade de teste do requisito levando em conta questões gerenciais, como custo, nivel de importância dos *stakeholders* e prazo de entrega. A depender dos resultados dos nóss pais, o status do requisito pode ser definido como sendo alto, médio ou baixo.

- **PrioridadeDeTesteDoRequisito:** O nó final da Rede Bayesiana, que vai definir qual vai ser a prioridade de teste do requisito dentre todos os outros. A probabilidade desse nó é resultado dos nós de StatusRequisito e RiscoDeBugsNoRequisito.

2.4 Abordagem Adaptativa para o Processo de Desenvolvimento do Software

A AADSP (*Adaptive Approach for Deployment of Software Process*) serve como um guia de processo de software criado pela *LABRASOFT* (Laboratório de Desenvolvimento de *Software*). Seu objetivo é propor uma processo com uma abordagem adaptativa para as micro e pequenas empresas (MPE) da cidade de Salvador. As práticas inovadoras que o AADSP propõe estão em conformidade com o modelo MPS-BR, da Softex, com práticas de métodos ágeis de desenvolvimento e com o PMBOK da PMI para gerenciamento de projetos [5] [27].

Os objetivos da AADSP são: promover maior qualidade dos softwares das MPEs (Médias e pequenas empresas), promover a geração de artefatos de documentos nos projetos de teste e promover práticas nunca antes vistas para ajudar as MPE no processo de software. Além disso, a AADSP promove a qualidade do software construído através do artefatos e documentos e, por último, traz adaptabilidade dos artefatos que compõem o sistema [5] [27].

A AADSP classifica os artefatos gerados durante o processo de software em três categorias: Essencial, Importante e Desejavel. Os artefatos são organizados de acordo com sua importância [5] [27].

Os artefatos essenciais são denominados artefatos-base, e são importantes para a AADSP; os artefatos importantes são consideráveis, mas sem obrigação de serem feitas, e a sua implementação adicionará os resultados adicionais ao processo. Já os artefatos desejáveis são os artefatos de pouca importância no processo de implantação ou entregam poucos resultados [5] [27].

Com base na guia MBR.BR, a AADSP listou seis gerências que tem como objetivo realizar o catálogo de artefatos que garantam a qualidade do processo [1]. Foras listadas as: Gerência de projetos, Gerência de requisitos e modelagem; Gerência de configuração e mudanças, Gerência de colaboradores, Gerência de testes e Gerência de Reutilização.

Este trabalho irá abranger a gerência de testes. Para esta gerência, os determinados artefatos são gerados:

- Plano de execução de testes do projeto (essencial)
- Lista de ocorrência de erros e não conformidades (importante)
- Lista de ações corretivas (importante)
- Controle de inspeção e qualidade (essencial)
- Termo de formação da Equipe de qualidade (desejável)
- Documento de homologação do teste de software (importante)
- Glossário de erros do projeto (importante)

3. TRABALHOS RELACIONADOS

Nessa seção, são apresentados e analisados os trabalhos feitos e apresentados na área de Otimização em Teste de Software, com enfoque em Priorização de Software. Existem diversos trabalhos que utilizam técnicas algoritmicas para a priorização de casos de testes, e um grupo de pesquisadores da Universidade Estadual do Ceará (UECE) realizam um estudo onde fizeram análises de trabalhos que plicam algoritmos metaheurísticos de otimização mono-objetivos e multiobjetivos para a otimização de teste de software.

Em Priorização de casos de teste, o grupo analisa cinco trabalhos. O trabalho de Rothermel et al. [28] é um dos mais iniciais sobre o tema [19]. Nele, algoritmos gulosos são utilizados para a priorização de casos de testes de regressão para o encontro de soluções ótimas. Os algoritmos são aplicados a partir de diferentes técnicas: são utilizados algoritmos que ordenavam casos de teste com base na sua cobertura de componentes de código, algoritmos que ordenam os casos de teste com base em cobertura de componentes de código não abordados anteriormente e algoritmos que ordenam casos de teste com base em sua capacidade estimada de revelar falhas nos componentes de código que eles cobrem.

No trabalho de Walcott et al.[29] é proposto um algoritmo genético para realização da priorização de casos de teste. A aplicação do algoritmo é comparada com algumas técnicas iniciais (casos executados na ordem normal e na ordem contrária). Como resultado, o algoritmo utilizado supera o algoritmo guloso [19].

O trabalho de Yoo e Harman[30] propõe uma abordagem para a priorização dos casos de teste utilizando o conceito de Pareto. Para a priorização dos casos de teste, é criado

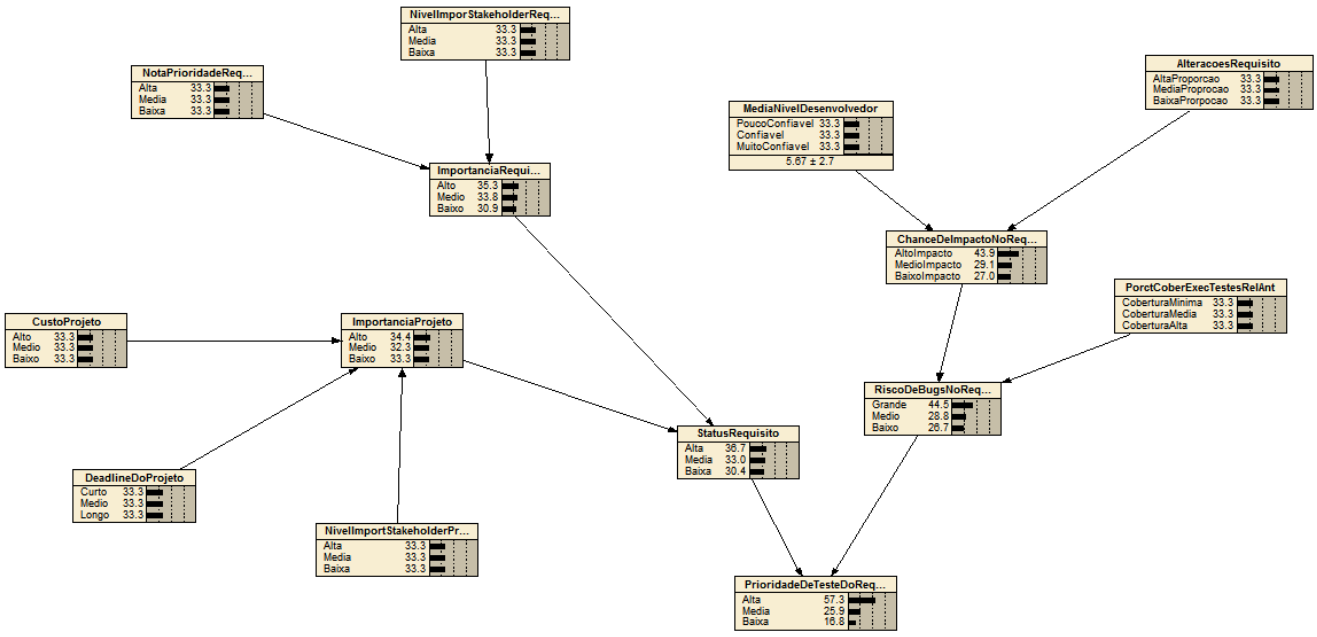

Figura 5: Representação da Rede Bayesiana proposta.

um vetor de variáveis para indicar a ordenação dos casos de testes [19].

Um trabalho de Maia et al.[31] utiliza o algoritmo de metaheurística GRASP Reativo para realizar a priorização dos testes. Além de ter sido executado com varios percentuais da *suite* de testes (1%, 2%, 3%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% e 100%) os resultados desse algoritmo são comparados a resultados de outros algoritmos anteriores, como algoritmos genéticos e algoritmos gulosos. O algoritmo GRASP Reativo apresenta o segundo melhor desempenho dentre os dois e possui o pior desempenho somente para o algoritmo guloso [19].

No trabalho de Li, Harman e Hierons [32] é feita uma comparação entre cinco algoritmos (Guloso, Guloso Adicional, Ótimo, *Hill-Climbing* e Genéticos). A análise usa 1000 *suites* de testes pequenas e 1000 *suites* de teste maiores. Foi detectado que, nas *suites* pequenas, o algoritmo genético é melhor que os demais e o guloso tem o pior desempenho; já na *suite* de teste maior os resultados são semelhantes aos das *suites* pequenas, mas com a diferença de que não há diferença significantes entre o algoritmo guloso e o algoritmo genético, do mesmo jeito que não há diferença significante entre o algoritmo Otimo,e o guloso adicional [19].

3.1 Comparativo entre os trabalhos

Como foi informado anteriormente, os trabalhos anteriores foram estudados por [19], e descrevem os trabalhos até agora conhecidos sobre priorização de casos de testes. O primeiro diferencial destacável que o *T-AADSP Test* terá é o de ter um módulo agregado para o registro de erros encontrados no sistema e de registro de casos de teste de um requisito. Além de ser útil para a equipe de testes devido a possibilidade de rastreamento e registro de *bugs*, esses registros ajudarão a calcular a nota dos desenvolvedores que, por sua vez, ajudarão a prever a probabilidade dos casos de testes cadastrados e indicará o nível de cobertura de testes em uma execução anterior. As ferramentas que foram usadas como comparação não estão acopladas a nenhum tipo de módulo de gerenciamento de *bugs*.

Outro diferencial será o fato de que o módulo de testes estará conectado a outro módulo de gerenciamento de requisitos. Isso faz com que os casos de testes sejam automaticamente vinculados aos requsitos do sistema, tornando fácil a visualização dos casos de testes prioritários a partir dos requisitos cadastrados no sistema.

A rede bayesiana foi escolhida para a priorização por permitir uma abordagem multiobjetiva, enquanto que os algoritmos analisados anteriormente são mono-objetivos. É necessário que a técnica de otimização seja multiobjetiva devido a existência de objetivos conflitantes; a Rede Bayesiana não irá priorizar os casos de testes somente baseado nos riscos que ele tem de encontrar erros, mas também levará em conta se o requisito que o caso de teste está vinculado é muito importante dentro do projeto (Um requisito cujo caso de teste tem risco alto de encontrar *bugs*, mas a sua importância não é grande não será de alta prioridade, por exemplo), o prazo de entrega de um projeto, o nível de importância do *stakeholder* vinculado ao requisito e o nível de experiência do programador.

4. A FERRAMENTA T-AADSP TEST

Como dito antes, a ferramenta *T-AADSP Test* é uma ferramenta que terá como objetivo realizar a predição dos casos de testes dos projetos cadastrados. Na construção do sistema, as seguintes etapas foram seguidas: Levantamento de requisitos, análise dos requisitos, a sua documentação, a implementação através de código e os testes. Os requisitos funcionais e nãi-funcionais, bem como os diagramas de caso de uso estão disponíveis no apêndice.

5. ASPECTOS DO DESENVOLVIMENTO DO SISTEMA

5.1 Arquitetura do Sistema

O padrão de arquitetura utilizado na construção do sistema foi o MVC (Model - Control - View). O MVC tem como objetivo dividir a aplicação em três camadas de modo que: a camada View (camada de interação com usuário) comunique-se com a camada Model (que disponibiliza os dados do sistema e também pode editá-los ou deletá-los) através da camada Controller (que recebe as solicitações do usuário pelo View e as redireciona para o Model)

5.2 Tecnologias Utilizadas

O T-AADSP Test utiliza as seguintes tecnologias:

- **JavaFX 11:** O JavaFX é uma biblioteca gráfica multimídia disponível para a linguagem Java que permite a construção de sistemas com interfaces gráficas com suporte a efeitos, imagens, sons e animações.
- **Apache POI 3.15:** O Apache POI é um *framework* disponível que permite a manipulação de de arquivos gerados na suíte Microsoft Office. Documentos de texto (.docx) e Planilhas (.xlsx) são alguns dos arquivos que podem ser manipulados.
- **JFoenix 9.0.8:** é uma biblioteca que permite customizar elementos presentes no JavaFX (Botões, Combobox, Áreas de texto) para o Google Material Design.
- **Font Awesome 8.1:** é uma blibioteca que disponibiliza um conjunto de fontes e icones através do CSS.
- **Jayes 2.0.1:** Jayes é uma API feita para o Java que permite a criação de Redes Bayesianas. o Jayes tem o seu código-fonte aberto, e é mantido pela Eclipse, através do seu programa de Code Recommenders.

5.3 Principais funcionalidades

O acesso ao T-AADSP Test é feito através da execução do arquivo .jar disponível, e o seu código fonte está disponível no GitHub [1]. Um video com o resumo das funcionalidades também pode ser encontralo no *YouTube*[2]. Se for a primeira vez que o sistema está sendo executado ou ele não localizar a planilha, o programa gera uma planilha em formato .xlsx automaticamente. Após o sistema carregar, um menu é exibido com todas as opções disponíveis.

Para fazer operações com um projeto (cadastro, edição, remoção ou visualização), é só clicar no menu "Projeto"que está no lado esquerdo do sistema. No cadastro ou edição de projeto, deve ser informado o nome do projeto, data de entrega em formato dd/MM/aaaa, o *stakeholder* responsável e o seu custo (em reais). Importante salientar que, se existir requisito vinculado ao projeto, o mesmo não pode ser deletado.

Depois do cadastro de um projeto, requisitos podem ser vinculados a ele. Clicando no menu "Requisito", os requisitos podem ser visualizados e editados ou excluídos, se o usuário desejar.

Para cadastrar um novo requisito, deve ser informado o nome do requisito, *stakeholder* responsável, o projeto o qual ele vai pertencer e a nota de prioridade dele para o projeto. Se existir caso de teste vinculado ao requisito ou se o requisito estiver sendo usado na *sprint* atual, o sistema não permite a exclusão do requisito.

Um *stakeholder* também pode ser criado, visualizado, excluído ou visualizado. Através do menu "*Stakeholder*", todas as operações ficam disponíveis.

Para o *stakeholder*, deve ser dado como entrada o nome e a nota de importância. Caso o *stakeholder* esteja vinculado a algum projeto ou a algum requisito, o sistema impede a sua exclusão.

O módulo de Plano de Teste permite que um caso de teste seja cadastrado a um requisito ou que os casos de testes do requisito sejam alterado, excluídos ou visualizados. Para cadastrar um novo caso de testes, deve ser informado a descrição do caso de teste e o tipo de teste que ele executa. Caso um caso de teste esteja sendo usado na *Sprint* atual e o usuário tente deletar, o sistema não permite.

Os desenvolvedores disponíveis para os projetos podem ser cadastrados da ferramenta. No menu "Desenvolvedores", os desenvolvedores ja cadastrados podem ser visualizados, podem ser editados, podem ser excluídos ou o usuário pode incluir novos.

Para tanto, deve ser informada o nome e o seu nível (Júnior, Pleno ou Sênior). O desenvolvedor também tem uma nota atrelada a ele, que é recalculada a cada *Sprint* mas, por motivos éticos, a ferramenta não exibe essa nota.

O T-AADSP permite o cadastro, visualização, edição e exclusão de *bugs*, que servirão para o calculo da nota do desenvolvedor. Para o *bug* ser cadastrado, deve ser informado o requisito aonde foi encontrado o *bug*, o título, o desenvolvedor responsável pelo *bug*, a descrição e o seu nível de impacto. Um *bug* pode ser cadastrado somente se uma *sprint* estiver aberta.

Uma *Sprint* delimita os requisitos que foram alterados e os casos de testes executados em um momento de tempo. o T-AADSP permite cadastrar facilmente uma *Sprint*, com sua data de inicio, data de fim e o seu nome.

Uma *sprint* pode ter até três estados: Pendente, Em andamento e finalizada: assim que uma *sprint* é criada, ela entra como pendente, ao selecionar uma *sprint* e iniciar ela, ela passa a ter o status em andamento.

Quando uma *Sprint* está em andamento, é possivel vincular o requisito alterado a *sprint* e indicar o status dos casos dos testes. Quando o usuário opta por vincular o requisito, ele deve informar o nível de impacto do requisito e os desenvolvedores que atuaram nele.

Quando o usuário decide informar o status dos casos de testes executado, primeiramente ele seleciona um requisito da *sprint*, e depois indica os status dos casos de testes desse requisito.

Antes de iniciar os testes da *sprint*, o usuário deve gerar a predição dos requisitos. No menu Predição, o usuário, após inserir os requisitos na *sprint*, pode gerar a lista de probabilidades, ordenando os requisitos a partir da rede bayesiana. Se não existir *sprint* em execução ou a predição já tiver sendo feita, o botão que permite a priorização fica bloqueado.

6. PROJETO EM ESTUDO

Para realizar o processo de avaliação da ferramenta, o primeiro passo realizado foi calibrar/treinar a rede bayesiana. Para isso, foram usados três cenários sintéticos, com dois requisitos cada.

[1]https://github.com/vitorssantana/T-AADSP-Test
[2]https://www.youtube.com/watch?v=3ETGsltNa1k

Como o objetivo da rede bayesiana na ferramenta é auxiliar a solução de conflitos acerca da prioridade de teste dos requisitos de vários projetos reais, foi realizado a avaliação com trinta requisitos de três projetos reais da empresa Computação Brasil. Para a predição, foram extraídos oito requisitos de um projeto de cartão combustível, oito do primeiro projeto e dois do segundo projeto de uma empresa pública de visualização de dados.

Os requisitos, os dados dos desenvolvedores e dos *stakeholders* foram cadastrados em conjunto com os casos de testes de cada requisito selecionado. Com isso, foi possível gerar a sugestão de ordem de execução dos testes dos requisitos utilizando todos os dados reais.

7. RESULTADOS E DISCUSSÕES

7.1 Validação usando dados sintéticos

7.1.1 Cenário 1

O **requisito A** tem as seguintes características:

- Participação de Desenvolvedores com nota média confiável (5.0)
- Baixa cobertura de Testes anteriores (30%)
- Impacto Alto das alterações realizadas (80%)
- Alto Nível de importância do requisito para o *stakeholder*
- *Stakeholder* com alto nível de importância para o projeto (requisito com nota 10)
- Alto Nível de importância do *stakeholder* envolvido no projeto (*stakeholder* com nota 8)
- Pertence a um Projeto com valor médio (R$45.000,00)
- Prazo de entrega médio (4 meses e meio a partir da data atual)

E o **requisito B** tem as seguintes características:

- Participação de Desenvolvedores com nota média muito confiável (8.3)
- Média cobertura de Testes anteriores (45%)
- Impacto Médio das alterações realizadas (50%)
- Médio Nível de importância do requisito para o *stakeholder* (nota 5.0)
- *Stakeholder* com médio nível de importância para o projeto (requisito com nota 6)
- Alto Nível de importância do *stakeholder* envolvido no projeto (nota 8.0)
- Pertence a um Projeto com valor baixo (R$15.000,00)
- Prazo de entrega curto (2 meses e meio a partir da data atual)

Após a entrada na rede, temos as seguintes probabilidades para cada requisito:

- Requisito A: (Alta: 73.8%, Média: 18.7%, Baixo: 7.48%)
- Requisito B: (Alta: 44.1%, Média: 36.9%, Baixo: 18.9%)

Como pode-se ver, os dois requisitos sairam com prioridade alta, mas o requisito A tem uma importância muitíssimo mais relevante do que o requisito B. Avaliando as entradas que calculam o risco de erros do requisito, nota-se que o **requisito A** tende muito mais a ter *bugs* devido ao alto impacto da alteração que sofreu na *sprint* atual (80%), e também os desenvolvedores responsáveis por essa alteração serem medianos, sendo reconhecidos apenas como confiáveis. Ao mesmo tempo, avalia-se que o **requisito B** não teve uma alteração tão impactante do que o **requisito A**, e os desenvolvedores responsáveis por essas alterações são considerados muito confiáveis, o que dá uma margem de confiança grande de que o que foi desenvolvido não terá tantos erros.

Além disso, no requisito A, a cobertura dos testes anteriores cobriu apenas 30% da funcionalidade, enquanto que no requisito B esse nível de cobertura conseguiu cobrir metade da funcionalidade. Isso significa dizer que tem muito mais chances de um *bug* não detectado na *sprint* anterior do requisito A ter passado e continuado na *Sprint* atual do que a *Sprint* B.

Dos dados de importância do requisito, nota-se que o requisito A é um requisito muito importante tanto para seu projeto, quanto para os *stakeholders*, enquanto o requisito B fica no meio termo: seus *stakehoders* tem importância mediana e a sua importância no projeto é de nível médio. Entretanto, nota-se que o projeto do requisito B é um projeto muito mais prioritário do que o do requisito A, visto que o mesmo tem uma data de entrega curtíssima, e os *stakeholders* que cuidam desse projeto são muito importantes. No requisito A o projeto também tem *stakeholders* muitíssimos importantes, mas é considerado de média importância devido a não ter uma data de entrega tão curta, apesar do seu custo médio, comparado com o custo baixo do projeto do requisito A.

Em resumo, podemos definir que, apesar de não ter um risco imediato de *bugs*, a importância alta do requisito B deve-se muito mais a condição que seu projeto está, com prazo curto e *stakeholders* de níveis altos. Apesar de estar em um projeto consideravelmente de importância média, o requisito A tem chances muito altas de *bugs* devido ao impacto sofrido durante as alterações.

7.1.2 Cenário 2

O **requisito C** tem as seguintes características:

- Participação de Desenvolvedores com nota média confiável (6.0)
- Média cobertura de Testes anteriores (55%)
- Impacto Médio das alterações realizadas (65%)
- Médio Nível de importância do requisito para o *stakeholder*
- Requisito com médio nível de importância para o projeto (requisito com nota 6)
- Médio Nível de importância do *stakeholder* envolvido no projeto (*stakeholder* com nota 5)
- Pertence a um Projeto com valor médio (R$50.000,00)
- Prazo de entrega curto (3 meses e meio a partir da data atual)

E o **requisito D** tem as seguintes características:

- Participação de Desenvolvedores com nota média confiável (5.0)
- Média cobertura de Testes anteriores (45%)
- Impacto baixo das alterações realizadas (35%)
- Médio Nível de importância do requisito para o *stakeholder* (nota 5.0)
- Requisito com baixo nível de importância para o projeto (requisito com nota 2)
- Alto Nível de importância do *stakeholder* envolvido no projeto (nota 8.0)
- Pertence a um Projeto com valor médio (R$65.000,00)
- Prazo de entrega médio (4 partir da data atual)

Após a entrada na rede, temos as seguintes probabilidades para cada requisito:

- Requisito C: (Alta: 28.9%, Média: 61.5%, Baixo: 9.6%)
- Requisito D: (Alta: 34.7%, Média: 42%, Baixo: 23.3%)

Comparando os dois requisitos, nota-se que eles são classificados como sendo de requisito de média prioridade de teste, so que o requisito C tem uma probabilidade maior com relação ao requisito D. Analisando os dados de entrada, percebe-se que o requisito D tem um risco menor de ter erros do que o requisito C porque as alterações sofridas no requisito D foram de muito menos impacto. Os dois foram alterados por desenvolvedores confiáveis e tiveram uma cobertura média de testes anteriores.

A importância do requisito C é media, porque a sua nota de importãncia dentro do projeto é considerada média, assim como a importância dos *stakeholders* vinculados a ele. Já o requisito D tem uma importância mais baixa porque a sua nota de prioridade dentro do seu próprio projeto é baixa.

Sobre o projeto, verifica-se que o requisito C está em um projeto considerado de média importância por causa do seu custo médio e o fato de que os *stakeholders* desse projeto também serem de média importância. Apesar de estar classificado como médio, o seu prazo de entraga é curto, que lhe da também uma probabilidade consideravelmente alta. O projeto do requisito D também é classificado como médio porque tanto o seu custo quanto o seu prazo de entrega são médios, mas o que eleva um pouco a sua probabilidade de ser alto é a alta importância que os *stakeholders* participantes do projeto tem.

7.1.3 Cenário 3

O **requisito E** tem as seguintes características:

- Participação de Desenvolvedores com nota média muito confiável (9.0)
- Alta cobertura de Testes anteriores (86%)
- Impacto Médio das alterações realizadas (65%)
- Médio Nível de importância do requisito para o *stakeholder*
- Requisito com médio nível de importância para o projeto (requisito com nota 5)
- Baixo Nível de importância do *stakeholder* envolvido no projeto (*stakeholder* com nota 2)
- Pertence a um Projeto com valor médio (R$45.000,00)
- Prazo de entrega longo (7 meses e meio a partir da data atual)

E o **requisito F** tem as seguintes características:

- Participação de Desenvolvedores com nota média muito confiável (8.0)
- Alta cobertura de Testes anteriores (75%)
- Impacto baixo das alterações realizadas (35%)
- Baixo Nível de importância do requisito para o *stakeholder* (nota 3.0)
- Requisito com baixo nível de importância para o projeto (requisito com nota 2)
- Alto Nível de importância do *stakeholder* envolvido no projeto (nota 8.0)
- Pertence a um Projeto com valor médio (R$55.000,00)
- Prazo de entrega médio (5 meses e meio a partir da data atual)

Após a entrada na rede, temos as seguintes probabilidades para cada requisito:

- Requisito E: (Alta: 23.9%, Média: 29%, Baixo: 47.2%)
- Requisito F: (Alta: 13.4%, Média: 26.2%, Baixo: 60.3%)

Os dois requisitos estão classificados como tendo prioridade baixa, mas nota-se que o requisito F tem uma probabilidade baixa maior que o requisito E. Essa diferença se dá primeiramente pelo risco que os dois apresentam de apresentar erros: enquanto o requisito E tem um risco um pouco mais elevado devido a alteração nele ter sido de médio impacto, o requisito F teve alteração de baixo impacto, o que significa que existe uma chance baixa de *bugs* surgirem. Os dois requisitos foram alterados por desenvolvedores muito confiáveis e tiveram cobertura alta nos testes anteriores.

O requisito F tem uma importância muito menor que o requisito E porque, mesmo os dois tendo nota de importância baixa, os *stakeholders* do requisito E são de média importância, e os do requisito F são de baixa importância. Ao analisar os projetos dos dois requisitos, nota-se uma situação contrária ao que se viu anteriormente: o projeto do requisito F tem uma importância maior em relação com o do requisito E devido ao seu prazo de entrega curto; o custo dos dois projetos são considerados médios e os *stakeholders* são considerados de baixa importância.

7.2 Validação usando dados reais

Utilizando os dados reais, a predição dos requisitos foram feitas em três *Sprints* separadas. Dessas três *sprints*, foram retiradas os cinco requisitos mais prioritários dentro do projeto.

7.2.1 *Sprint 1*

A primeira *Sprint* teve como entrada trinta requisitos dos três projetos diferentes e, da saída, foram extraídos os seguintes requisitos:

- Requisito: Atualizar Visitante
  - Nivel de impacto de alterações: Média proporção
  - Nível dos desenvolvedores: Pouco confiável
  - Porcentagem de cobertura dos testes anteriores: Cobertura mínima
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Média
  - Custo do projeto: Alto
  - Prazo de entrega do projeto: Médio
  - **Prioridade Alta: 74%,**
  - **Prioridade Media: 19%,**
  - **Prioridade Baixa: 7%**
- Requisito: [SDV024]
  - Nivel de impacto de alterações: Alta proporção
  - Nível dos desenvolvedores: Confiável
  - Porcentagem de cobertura dos testes anteriores: Cobertura mínima
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Média
  - Custo do projeto: Baixo
  - Prazo de entrega do projeto: Curto
  - **Prioridade Alta: 72%,**
  - **Prioridade Media: 18%,**
  - **Prioridade Baixa: 9%**
- Requisito: Consulta de compras do beneficiario
  - Nivel de impacto de alterações: Alta proporção
  - Nível dos desenvolvedores: Confiável
  - Porcentagem de cobertura dos testes anteriores: Cobertura mínima
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Alta
  - Custo do projeto: Médio
  - Prazo de entrega do projeto: Longo
  - **Prioridade Alta: 72%,**
  - **Prioridade Media: 18%,**
  - **Prioridade Baixa: 9%**
- Requisito: Controle de Acesso
  - Nivel de impacto de alterações: Alta proporção
  - Nível dos desenvolvedores: Muito confiável
  - Porcentagem de cobertura dos testes anteriores: Cobertura mínima
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Média
  - Custo do projeto: Alto
  - Prazo de entrega do projeto: Médio
  - **Prioridade Alta: 72%,**
  - **Prioridade Media: 21%,**
  - **Prioridade Baixa: 8%**
- Requisito: Cadastro de beneficiários em lote
  - Nivel de impacto de alterações: Alta proporção
  - Nível dos desenvolvedores: Muito confiável
  - Porcentagem de cobertura dos testes anteriores: Cobertura mínima
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Média
  - Custo do projeto: Médio
  - Prazo de entrega do projeto: Longo
  - **Prioridade Alta: 69%,**
  - **Prioridade Media: 21%,**
  - **Prioridade Baixa: 11%**

7.2.2 *Sprint 2*

Depois de indicar o status dos casos de testes de cada requisito e finalizada a *Sprint* 1, foi iniciada uma nova *Sprint*. Os seguintes requisitos foram escolhidos:

- Requisito: Login
  - Nivel de impacto de alterações: Alta proporcao
  - Nível dos desenvolvedores: Confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura alta
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Alta
  - Custo do projeto: Medio
  - Prazo de entrega do projeto: Longo

  - **Prioridade Alta: 60%,**
  - **Prioridade Media: 26%,**
  - **Prioridade Baixa: 14%**

- Requisito: ConsultarSCILicitacao
  - Nivel de impacto de alterações: Alta proporcao
  - Nível dos desenvolvedores: Pouco confiável
  - Porcentagem de cobertura dos testes anteriores: Cobertura média
  - Nivel de importância do *stakeholder* para o requisito: Baixa
  - Nota de prioridade do requisito: Media
  - Nivel de importância do *stakeholder* para o projeto: Media
  - Custo do projeto: Alto
  - Prazo de entrega do projeto: Medio
  - **Prioridade Alta: 55%,**
  - **Prioridade Media: 30%,**
  - **Prioridade Baixa: 15%**

- Requisito: Cadastrar Conselheiro
  - Nivel de impacto de alterações: Media proporcao
  - Nível dos desenvolvedores: Confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura média
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Media
  - Custo do projeto: Alto
  - Prazo de entrega do projeto: Medio
  - **Prioridade Alta: 55%,**
  - **Prioridade Media: 34%,**
  - **Prioridade Baixa: 11%**

- Requisito: Atualizar Conselheiro
  - Nivel de impacto de alterações: Media proporcao
  - Nível dos desenvolvedores: Confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura media
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Media
  - Custo do projeto: Alto
  - Prazo de entrega do projeto: Medio
  - **Prioridade Alta: 55%,**
  - **Prioridade Media: 34%,**
  - **Prioridade Baixa: 11%**

- Requisito: Editar convenio
  - Nivel de impacto de alterações: Media proporcao
  - Nível dos desenvolvedores: Muito confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura media
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Alta
  - Custo do projeto: Medio
  - Prazo de entrega do projeto: Longo
  - **Prioridade Alta: 54%,**
  - **Prioridade Media: 31%,**
  - **Prioridade Baixa: 15%**

7.2.3 Sprint 3

Depois de indicar os requisitos mais críticos da *Sprint* 2, indicar o status dos testes e finalizar a *Sprint* 2, foi gerado mais uma nova *Sprint*. O resultado da priorização da *Sprint* 3 foram:

- Requisito: Consultar Convênio
  - Nivel de impacto de alterações: Alta proporcao
  - Nível dos desenvolvedores: Confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura alta
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Media
  - Nivel de importância do *stakeholder* para o projeto: Alta
  - Custo do projeto: Medio
  - Prazo de entrega do projeto: Longo
  - **Prioridade Alta: 50%,**
  - **Prioridade Media: 30%,**
  - **Prioridade Baixa: 20%**

- Requisito: Movimentos por comerciante
  - Nivel de impacto de alterações: Média proporcao
  - Nível dos desenvolvedores: Confiável
  - Porcentagem de cobertura dos testes anteriores: Cobertura alta
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Alta
  - Custo do projeto: Média
  - Prazo de entrega do projeto: Longo

  - **Prioridade Alta: 49%,**
  - **Prioridade Media: 34%,**
  - **Prioridade Baixa: 17%**

- Requisito: Cadastrar endereço adm
  - Nivel de impacto de alterações: Alta proporcao
  - Nível dos desenvolvedores: Muito confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura média
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Baixa
  - Nivel de importância do *stakeholder* para o projeto: Alta
  - Custo do projeto: Medio
  - Prazo de entrega do projeto: Longo
  - **Prioridade Alta: 48%,**
  - **Prioridade Media: 29%,**
  - **Prioridade Baixa: 22%**

- Requisito: [SDV004]
  - Nivel de impacto de alterações: Alta proporcao
  - Nível dos desenvolvedores: Muito confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura media
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Baixa
  - Nivel de importância do *stakeholder* para o projeto: Media
  - Custo do projeto: Baixo
  - Prazo de entrega do projeto: Curto
  - **Prioridade Alta: 48%,**
  - **Prioridade Media: 29%,**
  - **Prioridade Baixa: 22%**

- Requisito: Bloquear Cartao
  - Nivel de impacto de alterações: Baixa proporcao
  - Nível dos desenvolvedores: Muito confiavel
  - Porcentagem de cobertura dos testes anteriores: Cobertura media
  - Nivel de importância do *stakeholder* para o requisito: Alta
  - Nota de prioridade do requisito: Alta
  - Nivel de importância do *stakeholder* para o projeto: Alta
  - Custo do projeto: Medio
  - Prazo de entrega do projeto: Longo
  - **Prioridade Alta: 47%,**
  - **Prioridade Media: 35%,**
  - **Prioridade Baixa: 18%**

8. CONCLUSÃO

Este trabalho apresentou a ferramenta T-AADSP Test, cujo propósito é gerenciar todo o processo de testes de acordo com a Abordagem Adaptativa para o Processo de Desenvolvimento do Software (AADSP). Além disso ele utiliza uma Rede Bayesiana integrada ao sistema para poder priorizar os casos de testes a serem testados.

Durante o estudo do tema com a pesquisa da fundamentação teórica, análise dos documentos de testes obtidos para a avaliação, treino da rede e contato com pessoas da área, percebeu-se que a priorização dos casos de teste de requisito é de uma importância muito grande. As empresas de software querem e precisam gerenciar melhor a etapa de testes dos seus projetos, sobretudo de empresas de pequeno e médio porte, que não constam um muito investimento de tempo e de pessoas.

Como trabalhos futuros, podem ser desenvolvidos módulos para a integração de outras gerências presentes no AADSP (como Gerência de projetos, Gerência de requisitos e modelagem, Gerência de configuração e mudanças, Gerência de colaboradores e Gerência de Reutilização). Esses novos módulos irão aprimorar a ferramenta tornando-a mais completa. Tal esforço permitirá que todos os requisitos propostos pela AADSP para um projeto se software sejam atendidos. Também, outros tipos de algoritmos para fazer estimativa dos casos de testes podem ser aplicados, como a utilização de Redes Neurais, Classificador Bayesiano ou Algoritmos Genáticos.

9. REFERÊNCIAS

[1] R. A. de Santana, "Eta - easy test automation: uma ferramenta para automação de testes funcionais web baseada em selenium webdriver e testng," tech. rep., Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA.

[2] H. L. M. de Meirelles Quintella and J. L. I. de Almeida, "Fábrica de software: análise do impacto na competitividade.," p. 13, 2006.

[3] L. do Carmo Caldas and A. M. da S. Pitangueira, "Tsuite: Uma ferramenta para seleção ótima de casos de teste manuais para regressão," tech. rep., Instituto Federal de Educação, Ciência e Tecnologia da Bahia, 2012.

[4] V. S. Luz, "Uma abordagem para automação de testes de aceitação utilizando selenium, testng e tabelas fit," tech. rep., Departamento de Computação e Automação do Centro de Tecnologia da Universidade Federal do Rio Grande do Norte, December 2012.

[5] LABRASOFT - Grupo de Pesquisa Laboratório de Desenvolvimento de Software, *AADSP Guia de implementação – Geral: Fundamentação para implantação da abordagem adaptativa para implantação de processo de software.*, 2016.

[6] L. do Carmo Caldas and A. M. da S. Pitangueira, "Uma ferramenta para selecionar casos de teste manuais de forma ótima," 10th International Conference on Information Systems and Technology Management – CONTECSI, March 2013.

[7] R. Bortoluci and M. Duduchi, "Um estudo de caso do processo de testes automáticos e manuais de software no desenvolvimento ágil," X WORKSHOP DE PÓS-GRADUAÇÃO E PESQUISA DO CENTRO

PAULA SOUZA, October 2015.
[8] L. R. P. Izabel, "Testes automatizados no processo de desenvolvimento de softwares," tech. rep., Universidade Federal do Rio de Janeiro, 2014.
[9] International Software Testing Qualifications Board, *Certified Tester - Foundation Level Syllabus*, 2011.
[10] P. C. Bernardo and F. Kon, "A importância dos testes automatizados - controle ágil, rápido e confiável de qualidade," *Engenharia de Software Magazine*, 2008.
[11] D. V. Pereira, "Estudo da ferramenta selenium ide para testes automatizados de aplicações web," tech. rep., Universidade Estadual de Maringá, 2012.
[12] "Os 13 principais tipos de testes de software!." https://targettrust.com.br/blog/os-13-principais-tipos-de-testes-de-software/. Accessed: 2018-08-02.
[13] P. C. Macedo, M. R. de Moreira, and R. d. C. Catini, "Uso de testes automatizados em projetos de software: Estudo da aplicação em software comercial," *Revista Universitas, Ano 5 - Nº 9*, december 2012.
[14] "Testes de software - entendendo defeitos, erros e falhas." https://www.devmedia.com.br/testes-de-software-entendendo-defeitos-erros-e-falhas/22280. Accessed: 2018-08-01.
[15] "Testes de software - conceitos básicos." https://testesw.wordpress.com/conceitos-basicos/. Accessed: 2018-08-01.
[16] "Conceitos de teste de software." https://www.desenvolvedormatteus.com.br/conceitos-testes-software/. Accessed: 2018-08-01.
[17] A. Neto and D. Neto, "Introdução a teste de software," 08 2018.
[18] F. Gomes De Freitas, C. Maia, B. Maia, D. Pinto Coutinho, G. Campos, and J. Souza, "Aplicação de metaheurísticas em problemas da engenharia de software: Revisão de literatura," 01 2009.
[19] F. G. de Freitas, C. L. B. Maia, G. A. L. de Campos, and J. T. de Souza, "Otimização em teste de software com aplicação de metaheurísticas," *Revista de Sistemas de*, vol. 5, pp. 3–13, 2010.
[20] P. R. A. Firmino, "Redes bayesianas para a parametrização da confiabilidade em sistemas complexos," 2004.
[21] A. Souza, "Redes bayesianas: Uma introdução aplicada a credit scoring," 2010.
[22] S. Pena, "Thomas bayes, o 'cara'!," *Revista Ciência Hoje, vol. 38, pp. 22-29*, 2006.
[23] H. Ziv and D. J. Richardson, "Constructing bayesian-network models of software testing and maintenance uncertainties," in *Software Maintenance, 1997. Proceedings., International Conference on*, pp. 100–109, IEEE, 1997.
[24] R. L. Marques and I. Dutra, "Redes bayesianas: o que são, para que servem, algoritmos e exemplos de aplicações," *Coppe Sistemas–Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil*, 2002.
[25] R. L. d. S. da Silva, *Um modelo de redes bayesianas aplicado a sistemas de realidade aumentada*. PhD thesis, Universidade Federal do Rio de Janeiro, 2006.
[26] D. Detoni, R. M. Araujo, and C. Cechinel, "Predição de reprovação de alunos de educação a distância utilizando contagem de interações," in *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 25, p. 896, 2014.
[27] A. C. S. Souza, C. Dias, and M. Macedo, *AADSP - Gerência de Requisitos*. São Paulo: Editora Ixtlan, 2018.
[28] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, pp. 929–948, Oct 2001.
[29] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Timeaware test suite prioritization," in *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, ISSTA '06, (New York, NY, USA), pp. 1–12, ACM, 2006.
[30] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ISSTA '07, (New York, NY, USA), pp. 140–150, ACM, 2007.
[31] C. L. B. Maia, R. A. F. do Carmo, F. G. de Freitas, G. A. L. de Campos, and J. T. de Souza, "Automated test case prioritization with reactive grasp," *Adv. Soft. Eng.*, vol. 2010, pp. 2:1–2:13, Jan. 2010.
[32] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, pp. 225–237, April 2007.

Apêndices

A. TABELA DE PROBABILIDADES DE CADA NÓ DA REDE BAYESIANA

MediaNivelDesenvolvedor	AlteracoesRequisito	AltoImpacto	MedioImpa...	BaixoImpa...
PoucoConfiavel	AltaProporcao	90	7	3
PoucoConfiavel	MediaProprocao	65	23	12
PoucoConfiavel	BaixaProrpocao	45	35	20
Confiavel	AltaProporcao	70	25	5
Confiavel	MediaProprocao	10	87	3
Confiavel	BaixaProrpocao	20	35	45
MuitoConfiavel	AltaProporcao	65	10	25
MuitoConfiavel	MediaProprocao	25	30	45
MuitoConfiavel	BaixaProrpocao	5	10	85

Figura 6: Tabela de Probabilidade do nó ChanceImpacto-Requisito

PorctCoberExecTestesRelAnt	ChanceDeImpactoNoRequisito	Grande	Medio	Baixo
CoberturaMinima	AltoImpacto	90	7	3
CoberturaMinima	MedioImpacto	50	30	20
CoberturaMinima	BaixoImpacto	25	30	45
CoberturaMedia	AltoImpacto	75	23	2
CoberturaMedia	MedioImpacto	7	90	3
CoberturaMedia	BaixoImpacto	15	35	50
CoberturaAlta	AltoImpacto	65	10	25
CoberturaAlta	MedioImpacto	15	50	35
CoberturaAlta	BaixoImpacto	3	7	90

Figura 7: Tabela de Probabilidade do nó RiscoDeBugsNo-Requisito

B. REQUISITOS DA FERRAMENTA

NivelImporStakeholderRequisito	NotaPrioridadeRequisito	Alto	Medio	Baixo
Alta	Alta	90	7	3
Alta	Media	45	45	10
Alta	Baixa	40	20	40
Media	Alta	50	40	10
Media	Media	0	100	0
Media	Baixa	25	35	40
Baixa	Alta	40	20	40
Baixa	Media	25	30	45
Baixa	Baixa	3	7	90

Figura 8: Tabela de Probabilidade do nó ImportanciaRequisito

NivelImportStakeholderProjeto	CustoProjeto	DeadlineDoProjeto	Alto	Medio	Baixo
Alta	Alto	Curto	100	0	0
Alta	Alto	Medio	70	25	5
Alta	Alto	Longo	60	10	30
Alta	Medio	Curto	75	20	5
Alta	Medio	Medio	25	65	10
Alta	Medio	Longo	30	40	30
Alta	Baixo	Curto	60	10	30
Alta	Baixo	Medio	30	30	40
Alta	Baixo	Longo	25	5	70
Media	Alto	Curto	68	28	4
Media	Alto	Medio	35	60	5
Media	Alto	Longo	30	30	40
Media	Medio	Curto	40	55	5
Media	Medio	Medio	10	85	5
Media	Medio	Longo	10	60	30
Media	Baixo	Curto	30	40	30
Media	Baixo	Medio	5	65	30
Media	Baixo	Longo	5	30	65
Baixa	Alto	Curto	65	5	30
Baixa	Alto	Medio	30	40	30
Baixa	Alto	Longo	25	10	65
Baixa	Medio	Curto	40	30	30
Baixa	Medio	Medio	10	65	25
Baixa	Medio	Longo	10	25	65
Baixa	Baixo	Curto	25	10	65
Baixa	Baixo	Medio	10	20	70
Baixa	Baixo	Longo	5	10	85

Figura 9: Tabela de Probabilidade do nó ImportanciaProjeto

ImportanciaRequisito	ImportanciaProjeto	Alta	Media	Baixa
Alto	Alto	95	5	0
Alto	Medio	75	20	5
Alto	Baixo	65	10	25
Medio	Alto	75	20	5
Medio	Medio	7	90	3
Medio	Baixo	5	40	55
Baixo	Alto	60	10	30
Baixo	Medio	5	55	40
Baixo	Baixo	0	5	95

Figura 10: Tabela de Probabilidade do nó StatusRequisito

StatusRequisito	RiscoDeBugsNoRequisito	Alta	Media	Baixa
Alta	Grande	95	5	0
Alta	Medio	65	25	10
Alta	Baixo	40	50	10
Media	Grande	60	35	5
Media	Medio	7	90	3
Media	Baixo	15	35	50
Baixa	Grande	25	25	50
Baixa	Medio	5	35	60
Baixa	Baixo	0	5	95

Figura 11: Tabela de Probabilidade do nó PrioridadeTesteDoRequisito

B.1 Requisitos Funcionais

Logo mais abaixo, são descritos alguns dos requisitos funcionais do sistema. Como existem muitos requisitos funcionais, apenas os principais estão sendo descritos. Os diagramas de caso de uso da aplicação são exibidas nas figuras 12 e 13.

B.1.1 Criação de novo projeto

O usuário poderá cadastrar os projetos de uma determinada empresa ou organização que estará utilizando o sistema. Os projetos serão cadastrados junto com o seu nome, o seu custo em reais, o *stakeholder* responsavel pelo projeto e o prazo de entrega com o formato dd/MM/aaaa.

B.1.2 Listagem e edição de projeto

O usuário poderá verificar todos os projetos cadastrados no sistema com todos os dados disponíveis, bem como poderá editar os dados de um projeto, se assim for disponível.

B.1.3 Criação de cadastro de novo desenvolvedor

O usuário poderá cadastrar todos os desenvolvedores que estarão participando dos desenvolvimentos dos projetos cadastrados. O desenvolvedor será cadastrado com o seu nome e com o seu nível de conhecimento (júnior, pleno e senior).

B.1.4 Listagem e edição de todos os desenvolvedores

O usuário poderá consultar o cadastro de todos os desenvolvedores cadastrados, e também poderá editar os dados dos desenvolvedores.

B.1.5 Criação de cadastro de novo Stakeholder

O usuário poderá criar os registros dos *stakeholders* participantes do projeto. Os *stakeholders* poderão estar vinculados aos projetos ou vinculado as tarefas. Os *stakeholders* serão cadastrados usando o seu nome e sua nota de importância (de 1 a 10).

B.1.6 Listagem e edição de todos os stakeholders disponíveis

Os *stakeholders* estarão disponíveis para consultas do usuário, assim como a alteração de seus dados (nome e nota de importância). A alteração das notas podem impactar nas predições das redes bayesianas.

B.1.7 Criação de requisitos

Os requisitos de um determinado projeto poderão ser cadastradas no sistema com os segundos dados: Titulo do requisito, Id do projeto que o requisito está vinculado, a nota de prioridade do requisito (de 1 a 10), e o Id do *stakeholder* que estará responsavel por esse requisito. Os requisitos terão um plano de testes vinculados para execução de testes e eles serão priorizados pela Rede Bayesiana.

B.1.8 Visualização dos requisitos cadastrados

Os requisitos poderão ser visualizados, assim como os seus dados, o seu *stakeholder* vinculado, e o projeto no qual ele está vinculado.

B.1.9 Criação de registro de defeitos

Os defeitos que irão surgir serão cadastradas no sistema, e a sua correção será entregue durante a *Sprint*. O defeito será cadastrada, com o seu titulo, a descrição desse defeito, o nivel de impacto, a sua prioridade dentro da lista, a tarefa ou o defeito que gerou o novo defeito (para que a nota do desenvolvedor seja recalculada) e o id do desenvolvedor que corrigirá esse defeito.

B.1.10 Visualizar lista de defeitos

Os defeitos cadastrados poderão ser visualizados no sistema, bem como os seus dados para alteração. Uma vez vinculados, a release do defeito não poderá ser mais mudada.

B.1.11 Criação de plano de teste por requisito

O usuário poderá inserir no sistema os casos de testes de um determinado requisito. Os casos de testes serão vinculados a um requisito e serão executados em cada *Sprint*.

B.1.12 Cadastro, Visualização e Edição de *Sprints*

O usuário poderá cadastrar uma *sprint* informando o seu nome, sua data de início e sua data de fim e o seu status (Pendente, Em Andamento ou Finalizada).

B.1.13 Iniciar *Sprint*

Quando não existe nenhuma *Sprint* do sistema em andamento, o usuário pode selecionar alguma pendente para poder iniciá-la. Depois de finalizada uma *Sprint* não pode ser iniciada novamente.

B.1.14 Finalizar *Sprint*

O usuário, depois que tiver gerado a predição, registrado todos os bugs e vinculado todas as alterações dos requisitos, poderá finalizar a *Sprint*. Uma vez finalizada, o sistema bloqueia a inserção de novos registros de *bugs* e de algum novo tipo de alteração nela.

B.1.15 Indicar os requisitos entregues em uma *Sprint*

Nas *Sprints* devem estar vinculadas os requisitos que foram modificados durante a sua execução. O usuário deve informar qual foi o nível de impacto das alterações (Baixo, Médio ou Alto).

B.1.16 Vincular os desenvolvedores que atuaram em um requisito dentro da *Sprint*

A Rede Bayesiana necessita saber o nível dos desenvolvedores que atuaram dentro de um requisito na *sprint* atual, para poderem verificar se as alterações feitas tem um risco menor ou maior de possuir *bugs*. O usuário escolherá o desenvolvedor e o seu percentual de participação na alteração do requisito (caso somente um desenvolvedor tenha atuado nesse requisito, o percentual de participação será de 100/

B.1.17 Gerar predição dos casos de testes

A geração da predição levará em conta os dados das tarefas dos *bugs* de todos os requisitos e *stakeholders* vinculados para indicar qual requisito terá prioridade de ser testado. Basicamente a predição levará em conta a chance que o requisito tem de encontar defeitos X a importância do requisito para o projeto e para os *stakeholders*.

B.1.18 Indicar status dos casos de testes executados na *Sprint*

Depois da predição e da execução dos testes, o usuário indicará o status final do teste: NOK (foi achado erro durante a execução ou o teste não pôde ser executado), N/A (A execução não é necessáriao no momento) e OK (O teste foi executado com sucesso).

B.1.19 Finalizar *Sprint*

Depois da entrega dos testes, o usuário poderá finalizar a *sprint*. Uma vez finalizada, os dados da *sprint* não poderão mais ser alterados e a próxima *sprint* terá que ser iniciada manualmente.

B.1.20 Recalcular Nota do Desenvolvedor

Quando a *Sprint* é finalizada, os *bugs* que foram encontrados servirão para recalcular a nota dos desenvolvedores. Quanto mais bugs forem registrados no nome daquele desenvolvedor, maior será o decrésimo na sua nota. Caso não tenha sido detectado nehum bug, o sistema irá aumentar a nota desse desenvolvedor.

B.2 Requisitos Não-Funcionais

B.2.1 Usabilidade

O sistema deve ser fácil de usar, assim como as informações devem estar facilmente acessíveis para o usuário através de uma lista com tudo o que foi cadastrado e ter um design agradável para quem está operando o sistema no momento.

B.2.2 Compatibilidade

O sistema deve rodar satisfatoriamente em quaisquer sistemas operacionais desktop existentes atualmente. Para isso, é necessário somente ter o *Java Runtime Environment* (JRE) instalado no respectivo sistema.

B.2.3 Portabilidade

O sistema deve ser leve, para que o usuário o consiga levar facilmente o armazenando, por exemplo, eum um *pen-drive*, em alguma solução *cloud* ou em outro meio possível de armazenamento digital.

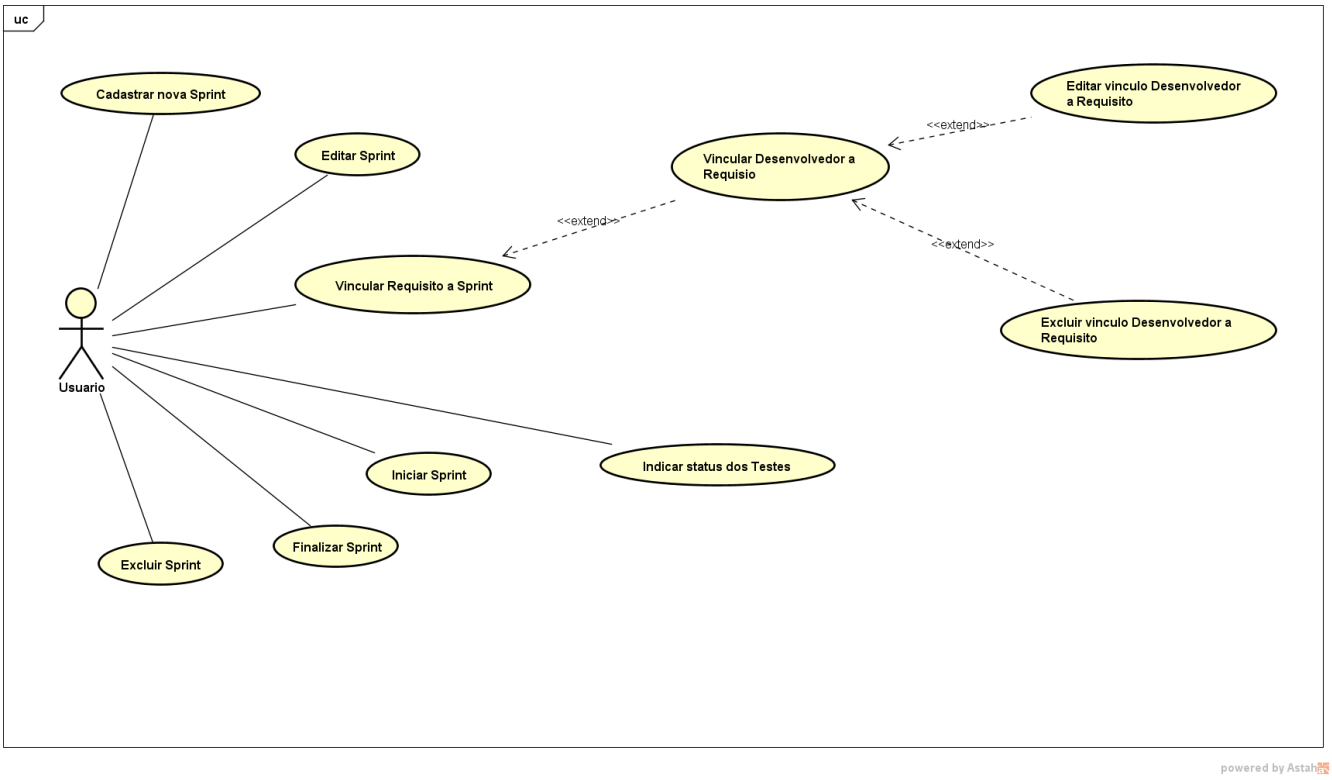

Figura 12: Diagrama de caso de uso que representa todas as funcionalidades vinculadas a *Sprint*

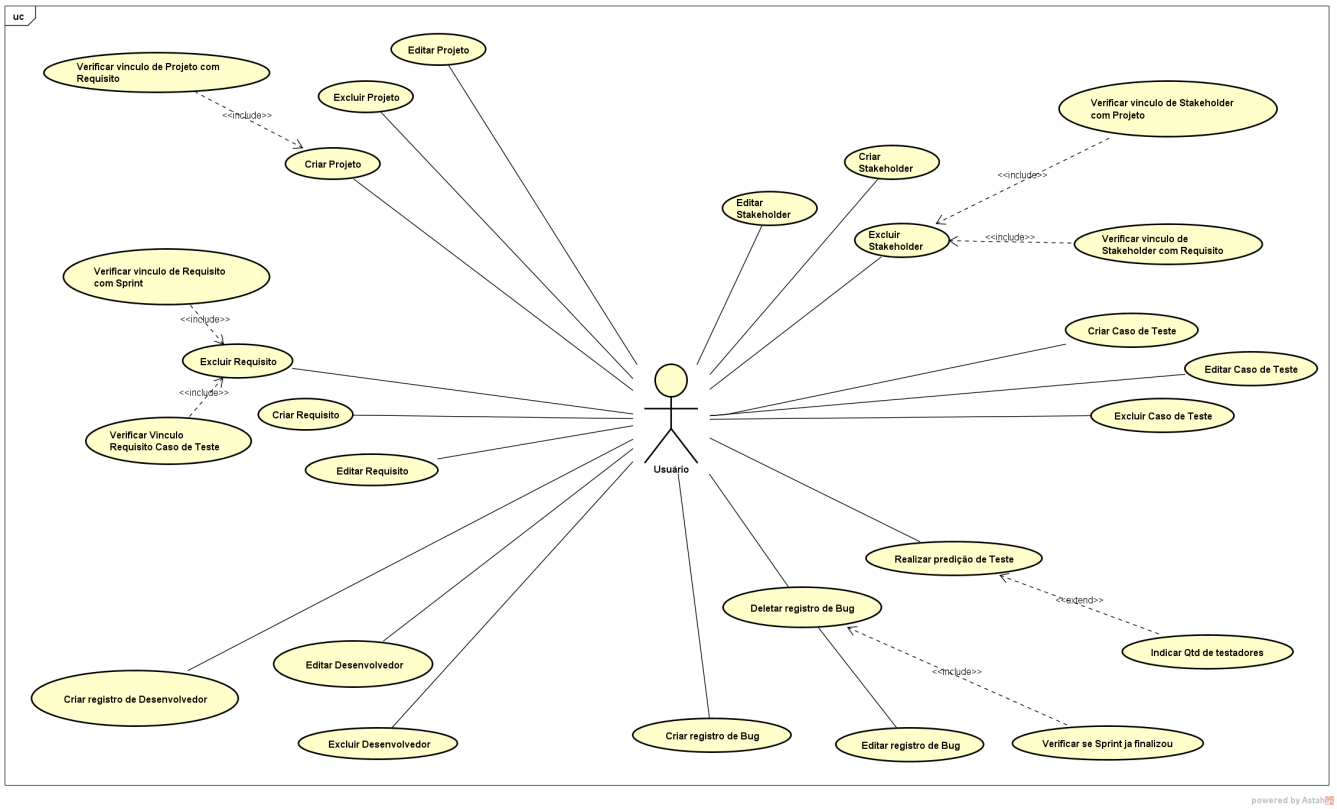

Figura 13: Diagrama de caso de uso das funcionalidades restantes.

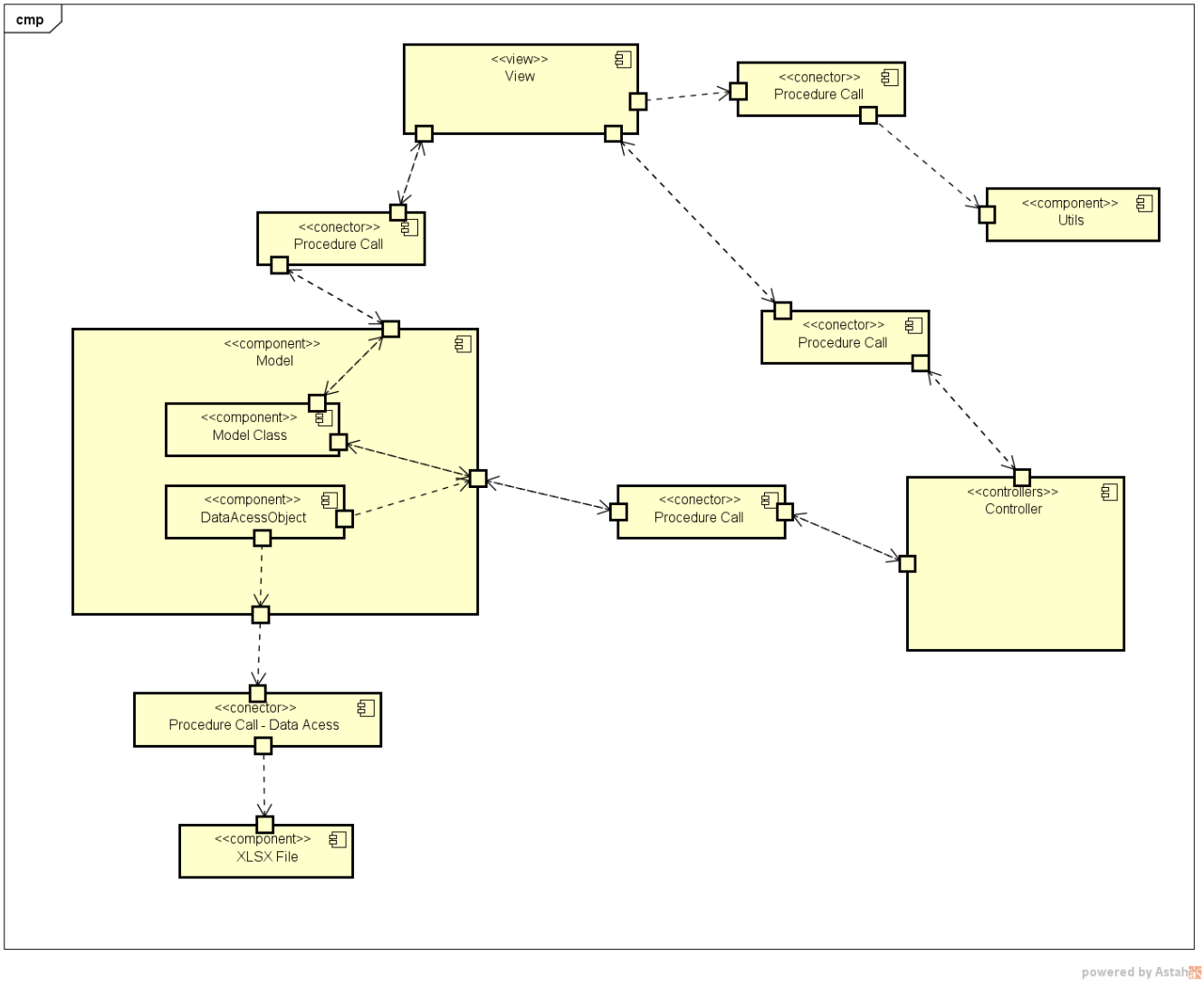

Figura 14: Diagrama de modelagem arquitetural do sistema.