Antonio Carlos dos Santos Souza
Vitor Souza Silva Santana
Hugo Saba Pereira Cardoso
Aline de Oliveira Machado
Bruno Santos Oliveira
Thalles Caribé Santiago Silva

Redes Bayesianas: AADSP - Gerência de Teste

1° Edição

São José dos Pinhais
BRAZILIAN JOURNALS PUBLICAÇÕES DE PERIÓDICOS E EDITORA
2020

Redes Bayesianas:
AADSP - Gerência de Teste

1° Edição

Brazilian Journals Editora
2020

Editora Executiva: Barbara Luzia Sartor Bonfim Catapan
Diagramação: Sabrina Binotti
Edição de Arte: Sabrina Binotti
Revisão: Prof. Dr. Luiz Machado dos Santos
e Prof. Dr. Ronaldo Pedreira Silva

Dados Internacionais de Catalogação na Publicação

S729r Souza, Antonio Carlos Santos

Redes bayesianas: AADSP - gerência de teste / Antonio Carlos Santos Souza. São José dos Pinhais: Editora Brazilian Journals, 2020.
52 p.

Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
Inclui: Bibliografia
ISBN: 978-65-86230-02-4

1. *Software*. 2. Rede Bayesiana.
I. Souza, Antonio Carlos Santos. II. Título.

Brazilian Journals Editora
São José dos Pinhais – Paraná – Brasil
www.brazilianjournals.com.br
editora@brazilianjournals.com.br

AUTORES

Antonio Carlos dos Santos Souza
Doutor em Ciência da Computação UFBA, Mestre em Modelagem Computacional FVC, Bacharel em Informática UCSAL e Técnico em Instrumentação Industrial na ETFBa. Docente efetivo do IFBA.
E-mail: antoniocarlos@ifba.edu.br

Vitor Souza Silva Santana
Graduado do curso tecnológico de Análise e Desenvolvimento de Sistemas do IFBA e analista de testes de uma empresa da área de medicina nuclear.
E-mail: vitorsouzassantana@gmail.com

Hugo Saba Pereira Cardoso
Doutor em Difusão do Conhecimento UFBA, Mestre em Modelagem Computacional pela FVC, Especialização em Computação Científica pela Fundação Visconde de Cairu (FVC) e Graduação em Processamento de Dados pela Faculdade Rui Barbosa. Professor Efetivo da UNEB.
E-mail: hugosaba@gmail.com

Aline de Oliveira Machado
Bacharel em Ciências Biológicas pela UFBA (2010) e Graduada em Análise e Desenvolvimento de Sistemas pelo IFBA (2018). Tem experiência com Stock Market, Redes Neurais Artificiais, Algoritmos Genéticos e Saúde Pública. Atualmente é mestranda em engenharia de sistemas e produtos no IFBA.
E-mail: alinemach@gmail.com

Bruno Santos Oliveira
Sócio-Diretor da Computação Brasil, bacharel em análise de sistemas pela UNEB e especialista em IA e Negócios.
E-mail: bruno.oliveira@computacaobrasil.com.br

Thalles Caribé Santiago Silva
Sócio-Diretor da Inovatic Serviços em Informática e palestrante na área de redes e tecnologia.
E-mail: thallescsantiago@gmail.com

DEDICATÓRIA

Dedicado a Deus.

AGRADECIMENTOS

À Pro-reitoria de Pesquisa, Pós-Graduação e Inovação do Instituto Federal da Bahia pelo apoio financeiro ao projeto "Uma abordagem adaptativa para implantação de processos de desenvolvimento de *software* em micro e pequenas empresas de Salvador/BA" aprovado no edital 05/2014/PRPGI/IFBA.

SUMÁRIO

DOI 10.35587/brj.ed.0000102

PREFÁCIO

Diversas abordagens trabalham de forma relevante para a melhoria do processo de *software* e organizam disciplinas na área de projetos de *software,* ofertando à indústria de *software* várias possibilidades de metodologias, técnicas e ferramentas. No contexto de uma empresa de desenvolvimento de *software,* em geral, os analistas e programadores não lidam no dia a dia com as metodologias formais de processo de *software Teste*

O desejo de confecção dessa série de livros é a difusão do conhecimento representado na abordagem *AADSP - Adaptive Approach for Deployment of Software Process* - A abordagem Adaptativa para Implantação de Processo de Software, proposta pelo Labrasoft - Laboratório de Desenvolvimento de *Software* - e financiado desde 2014 pelo edital de fortalecimento de pesquisa da Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação do IFBA - Instituto Federal da Bahia. Para tanto, foram consideradas a dimensão epistemológica e a dimensão ontológica, nos seguintes desafios: Bem-estar e realização das pessoas envolvidas; Transparência e melhoria da comunicação com o cliente ou consumidor, usando um sistema de representação do conhecimento; Pré-avaliação dos projetos da organização com base em artefato; Melhoria contínua do retorno por investimento.

ABRANGÊNCIA

O primeiro destinatário desta obra é o testador de *software* ou analistas de testes das empresas de desenvolvimento de *software.*

REDES BAYESIANAS

No livro, abordaremos a modelagem estocástica e o uso das redes bayesianas na predição ou priorização da execução dos testes dos requisitos de vários projetos, representando o Conhecimento Exato x Conhecimento Incerto presentes nas bases de conhecimento tácito ou até mesmo no conhecimento explícito.

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 [7]. 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 [8].

Diante desse mercado competitivo, as indústrias de *software* estão em busca de métodos e processos que façam com que a produtividade dos seus projetos aumente [7]. 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 [9].

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 [10]. 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 [10].

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 [7].

Para tentar minimizar este problema, a abordagem AADSP, que é uma abordagem para implantação de processo de *software* adaptativa criada pelo grupo de pesquisa do IFBA LABRASOFT, permite o gerenciamento do módulo de teste dos projetos de *softwares* de uma empresa usando as redes bayesianas e um sistema de representação do conhecimento. [11].

O *AADSP* possibilita 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. A abordagem considera os 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, a abordagem faz uso de um algoritmo de Redes Bayesianas para calcular o risco de *bugs* de um módulo. A AADSP usa 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, em um modelo estocástico.

O resultado da predição com um alto índice de acerto nas escolhas dos casos de teste faz com que na sua aplicação em projetos de desenvolvimento reais auxilie a equipe de testes a reduzir o tempo gasto na escolha dos casos de testes mais prioritários. Isto é, escolhendo os casos de testes que têm mais chances de capturarem erros, reduzindo, consequentemente, o custo da etapa de testes [12].

CAPÍTULO 01

REFERENCIAL TEÓRICO

1. TESTE DE SOFTWARE E OTIMIZAÇÃO DE TESTE

1.1 TESTE DE SOFTWARE

O teste é uma investigação feita geralmente por um testador, em um *software* em fase de desenvolvimento, geralmente por um testador, de um *software*, em fase de desenvolvimento [7]. No teste geralmente se busca a presença e o reporte dos defeitos e falhas encontrados na aplicação [12] [13], 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 [12] [14] [15].

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 metodologias ágeis) [16]. Enquanto isso, a realização dos testes depois da implementação traz problemas por causa dos inúmeros erros acumulados devido a falhas de testadores (ação humana é suscetível a erros) [14]. A aplicação do teste durante todo o projeto previne a ocorrência de muitos erros, já que a qualidade está sendo controlada a todo tempo [12].

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) [14]. Como essas falhas podem levar a prejuízos financeiros ou até a integridade de pessoas, é extremamente necessário realizar testes em *softwares* [12].

É impossível testar um *software* por completo, pois não há como garantir que todas as combinações do sistema foram testadas [7]. 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 [9]. 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 [17]. Um defeito encontrado durante a criação do *design* do sistema e no requisito custa 10 vezes menos do que encontrado durante a codificação e 1000 menos do que encontrado no cliente [14].

As atividades de teste estão divididas em duas técnicas, quatro tipos e cinco níveis [7]. As técnicas de teste têm como objetivo aferir o funcionamento do sistema internamente ou externamente [17]. Já os tipos de teste têm a finalidade de testar o *software* através de um objetivo particular [18]. Os níveis dividem o teste de um *software* em camadas [19].

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*) [14] [15] [10]. Na hierarquia dos termos, o erro produz um defeito, que por sua vez resulta em falhas no software [15].

O erro é provocado por alguma ação humana que produz um resultado incorreto. O erro pode ser causado, por exemplo, por uma falha de programação em algum trecho de código ou por uma falha na análise de especificação de requisitos [14] [20].

1.1.1 NÍVEIS, TÉCNICAS E TIPOS DE TESTES

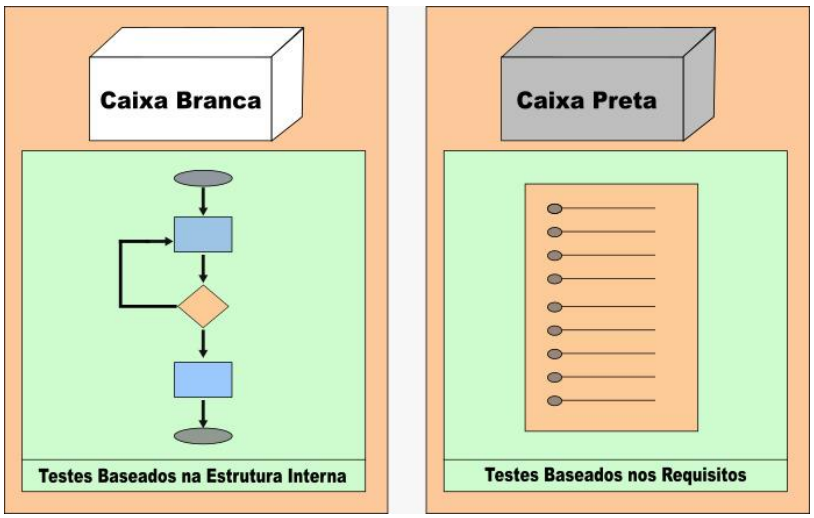
Técnicas de teste

Como mencionado anteriormente, existem duas técnicas de execução de testes: testes de caixa preta e caixa branca [10]. Como mostra a figura 2.1, as duas tem o objetivo de avaliar o funcionamento do código de diferentes pontos 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 [14]. 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 [15] [14] [10].

Figura 2.1: Representação visual da estrutura de um teste de caixa-branca e de caixa-preta [1].



Tipos de Testes

Existem quatro tipos de testes que podem ser aplicados em um software: Funcionais, Não-Funcionais, Estruturais e de Manutenção [7]. Cada tipo de teste tem como objetivo realizar o teste de partes distintas de um projeto [10].

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 [14].

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 [14].

Os Testes Não-Funcionais são realizados em um *software* para se obter a resposta desse *software* em variadas situações [9]. 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 [14].

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 [7].

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 [7]. 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 [14] [10].

Níveis de Testes

Os testes estão divididos em cinco níveis [7], 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 [19]. Os níveis de testes podem ser unitários, de integração, de validação, de sistema e de aceitação [14] [21].

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 [21]. Geralmente, o próprio desenvolvedor do módulo que contém as unidades realiza os testes [14].

No teste de integração, deve-se testar a interação entre os diferentes módulos do sistema [10]. O teste de integração visa buscar consequentes erros associados a interação de dois módulos [21]. 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 [15].

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 [7].

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 [10].

Nessa etapa, os testadores fazem baterias de testes caixa-preta para verificar o comportamento do sistema a cada caso de uso final [14].

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 [10]. 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 [7].

1.1.2 MEDOTOLOGIAS DE DESENVOLVIMENTO ORIENTADAS A TESTE

TDD *(Test Driven Development)*

Estudos e proposições foram feitos para adaptar as metodologias de desenvolvimento de *software* a área de testes. O principal objetivo era promover uma melhor qualidade do *software.*

A partir dessas proposições, metodologias e técnicas de desenvolvimento que priorizasse o teste de *software* foram propostas, com o intuito de que ele fosse o elemento central do desenvolvimento do *software* e desse jeito, melhorar a qualidade do que é entregue ao cliente final [22]. Nesse contexto, surgiu o TDD *(Test Driven Development).* O TDD é uma metodologia de desenvolvimento de *software* que tem como principal interesse fazer com que os testes sejam desenvolvidos antes do desenvolvimento, promovendo uma maior qualidade do *software* a ser entregue. O objetivo é ter um sistema cujo código seja facilmente testável, que seja de fácil manutenção e de fácil evolução [22].

O TDD propõe que, dentro do desenvolvimento de *software,* os próprios desenvolvedores façam o desenvolvimento dos testes unitários [23] [22]. Basicamente, o desenvolvedor deve desenvolver uma classe de teste para cada bloco de trecho de código escrito. Esses testes são automatizados, na mesma linguagem que o código que o desenvolvedor está codando, e a sua rápida execução permite que eventuais erros de codificação ou eventuais impactos do código em outros trechos do sistema sejam facilmente identificados [24] [23].

Além disso, o TDD força o desenvolvedor a uma escrita código de mais qualidade, pois o código do mesmo será reavaliado a todo momento através da execução dos testes, e ajuda o programador a pensar na solução mais simples para um problema, por que os testes escritos o ajudarão a pensar no que o código deve fazer, em vez de focar diretamente na implementação [24]. Existem *frameworks* de

testes unitários para várias linguagens de programação, como JUnit (Java)[1], NUnit (C)[2], xUnit (.NET Framework)[3] e PHPUnit (PHP)[3]. Segundo seu criador, Kent Beck, o TDD faz com que desenvolvedores e designers de código façam códigos simples e que inspirem confiança [25].

BDD *(Behaviour Driven Development)*

O BDD pode ser definido como sendo uma metodologia de desenvolvimento de *software* cujo fator central é o comportamento de uma funcionalidade [25]. O BDD é considerado uma evolução do TDD, devido ao fato de que os testes ainda são escritos antes do desenvolvimento da funcionalidade; a diferença é que os testes descreverão a funcionalidade que está sendo desenvolvida, guiando assim o desenvolvimento por parte dos desenvolvedores [25].

No BDD existe o conceito da estória de usuário, que basicamente descreve uma funcionalidade e a regra de negócio que será desenvolvida [25]. As estórias podem ser fornecidas pelo cliente ou retiradas da documentação do sistema, e devem ser de fácil entendimento para que o desenvolvedor às entendam sem nenhuma ambiguidade [25]. Para facilitar esse entendimento, as estórias são escritas usando uma linguagem natural juntamente com linguagens oblíquas, para que todos os envolvidos no projeto (cliente, coordenador, desenvolvedor, testadores) possam entender e escrever tais estórias [25]. Geralmente, uma estória de usuário tem a seguinte estrutura: 1. Dado... 2. E... 3. Quando... 4. Então...

Como exemplo, podemos descrever facilmente uma estória de usuário para acessar o *gmail* estando na página inicial do Google: 1. Dado que eu esteja com meu browser aberto; 2. E eu entre no site www.google.com; 3. Quando eu clicar no link *"Gmail";* 4. Então o *browser* deve carregar minha página oficial do *gmail.*

Como se percebe, o BDD expõe de maneira sucinta e simples qual vai ser o comportamento de determinada funcionalidade a tal ação, facilitando o entendimento de todas as pessoas participantes do projeto sobre determinada funcionalidade, e assim permitir um maior alinhamento e facilitando a comunicação entre essas pessoas. A aplicação do BDD em uma equipe de desenvolvimento de *software* tem as seguintes vantagens [23][25]: Facilidade de entendimento de toda a equipe: todos irão

[1] https://junit.org/junit5/

[2] https://nunit.org [3]https://xunit.net

[3] https://phpunit.de

se guiar pelas estórias de usuário, fazendo com que os objetivos do projeto seja conhecido por todos; Maior clareza para o desenvolvedor do que deve ser feito: as estórias farão com que o desenvolvedor foque apenas em desenvolver o comportamento correto do *software;* Permitir o desenvolvimento de testes automatizados de regressão e de integração: testes automatizados conseguem verificar rapidamente se há erro na implementação da regra ou se o impacto da alteração atingiu outras funcionalidades; As estórias podem facilmente servir de documentação: em uma metodologia ágil, não é recomendado que se faça uma documentação extensa, pois esse processo é custoso. As estórias podem servir muito bem como uma documentação oficial do sistema, pois a sua escrita não impacta em custo financeiro e de tempo para o projeto, além de ser de fácil entendimento.

Existem vários *frameworks* que permitem a automatização das estórias de usuário. O Cucumber[4], é um dos *frameworks* mais utilizados para escrita das estórias de usuário e permite integração com API's de automação, como o Selenium WebDriver[5]. Além disso, está disponível para várias linguagens de programação, como Java, Ruby, C++ ou JS. O RSpec[6] também permite a escrita de estórias, só que ao contrário do Cucumber, ele á disponível somente para a linguagem Ruby. Permite integração com Selenium Webdriver ou com o Capybara [7] (API de automação disponível somente para Ruby).

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 [26].

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 [27]. Esses problemas podem

[4] https://cucumber.io

[5] https://www.seleniumhq.org/projects/webdriver/

[6] https://rspec.info

[7] https://github.com/teamcapybara/capybara

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 [27].

Com a união da área de otimização matemática com a área de engenharia de *software,* surgiu uma nova área de pesquisa, denominada de Otimização em Engenharia de *software* ou, em inglês, *Search-based Software Engineering* (SBSE) [27] [26]. 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* [27] [26].

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 têm como objetivo otimizar algum processo da etapa de Teste de *Software.* Conforme a Tabela 2.1, 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 [27].

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 possível de testes no sistema mesmo com a sua parada [27] [26]. Na tabela 2.2 podem ser vistas todas as áreas que tiveram algoritmos implementados para a otimização.

Tabela 2.1: Áreas já pesquisadas da área de otimização de Engenharia de *Software* [27].

Áreas 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 de *Heap*
	Tamanho de Código
Projeto de *Software*	Modularização

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 Engenharia de *Software* tradicional não conseguiam, ou conseguiam, mas de forma insatisfatória [27].

Tabela 2.2: Problemas atacados na área de Otimização em Teste de *Software* [27].

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

3. REDES BAYESIANAS

No mundo real, existem decisões que exigem a análise de múltiplos fatores. Fatores esses que podem afetar significativamente o resultado de uma decisão tomada. Para auxiliar na análise de fatores nas diversas áreas humanas (saúde, educação, etc), a área da Inteligência Artificial (IA) desenvolveu técnicas de algoritmos que conseguem se adaptar a várias situações, como diagnose médica, reconhecimento de padrões ou análise química.

Uma das técnicas de IA (Inteligência Artificial) e de representação do conhecimento que se enquadra em um modelo estocástico faz uso das *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 [28].

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 [29]. 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.

O renomado reverendo inglês Thomas Bayes (1701-1776) foi o idealizador do chamado teorema *bayesiano,* teorema esse que é a base das *Redes Bayesianas* [30]. 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 de o evento acontecer ou não (chamado de probabilidade a posteriori) [30].

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 [29].

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 [31]. 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 [29].

Teoria de Bayes

A teoria de Bayes é a base de todo o funcionamento de uma *Rede Bayesiana.* Ela permite extrair uma igualdade simples da probabilidade de Prob (A/B) e de Prob (B/A) através da fórmula:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Onde o resultado será a probabilidade total. Através do cálculo utilizando probabilidades a priori (Prob A), é possível obter a probabilidade a posteriori (Prob A|B) de eventos e assim, permitir a erificação da probabilidade [32].

A partir de sua aplicação, o Teorema de Bayes permite que se obtenha uma inferência somente a partir dos dados fornecidos anteriormente (a priori), ao invés de utilizar os dados que fossem obtidos sobre toda informação relevante do evento. Junto com a possibilidade de interpretação de probabilidade como crença, o Teorema de Bayes no final fornece a probabilidade dos parâmetros fornecidos no modelo estatístico [32].

Para ficar mais claro, eis um exemplo do teorema aplicado na área médica [33]: A meningite causa torcicolo em 50 % dos casos, mas também se sabe que um caso de meningite atinge 1 em cada 50.000 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:

$$P(M \mid T) = \frac{P(T \mid M)\, P(M)}{P(T)}$$

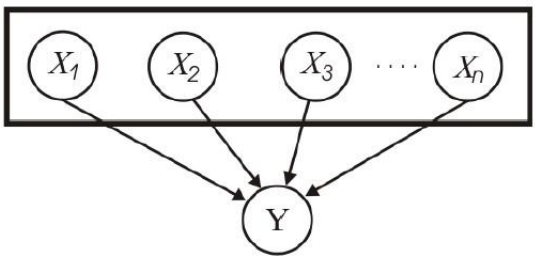
- **(0.5 x 1/50000)/(1/20) = 0.0002**

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

Funcionamento

Uma *Rede Bayesiana* é constituída de representações gráficas dos relacionamentos entre variáveis, [29] cuja representação pode ser vista na figura 3.2, e cada relacionamento representa uma dependência de causa entre as variáveis conectadas [2]. 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 3.1 mostra as representações gráficas de um nó e de um arco.

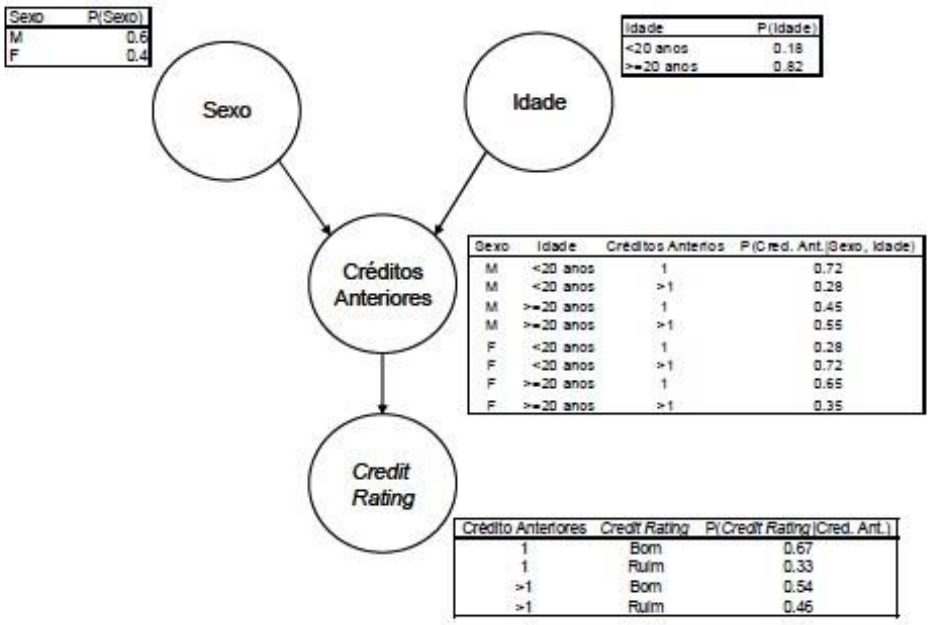
Figura 3.1: Exemplo de uma *Rede Bayesiana* [2].



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 [29][33][2].

Uma vez definida a rede, é necessário definir a tabela de probabilidades para cada nó [33]. Nessa tabela, fica registrada a probabilidade condicional do nó filho ligada diretamente aos valores inseridos no(s) nó(s) pai(s) [33][29].

Figura 3.2: Exemplo de *Rede Bayesiana de Credit Score*, com as condicionais já definidas dos nós filhos (Créditos Anteriores e *Credit Rating)* [2].



Uso das Redes Bayesianas

Existem vários exemplos de uso de Redes Bayesianas nas áreas do conhecimento humano, principalmente na área de diagnóstico [33]. 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 [2]. Para isso, ele utilizou três características: cor, forma e textura [2].

Previsão do fluxo de tráfego O estudo proposto por [34] realiza uma previsão do fluxo de tráfego entre linhas rodoviárias a partir de Rede Bayesiana. A rede é formada por nós de causa e nós de efeito. Sendo que os primeiros referem-se aos dados utilizados para previsão, enquanto os segundos são os dados a serem previstos. Com base no cálculo do erro quadrático médio a previsão do fluxo de tráfego é efetuada, [34] afirmam que essa previsão é eficaz mesmo que os dados a serem dispostos na rede sejam incompletos, o que confere robustez à tal abordagem.

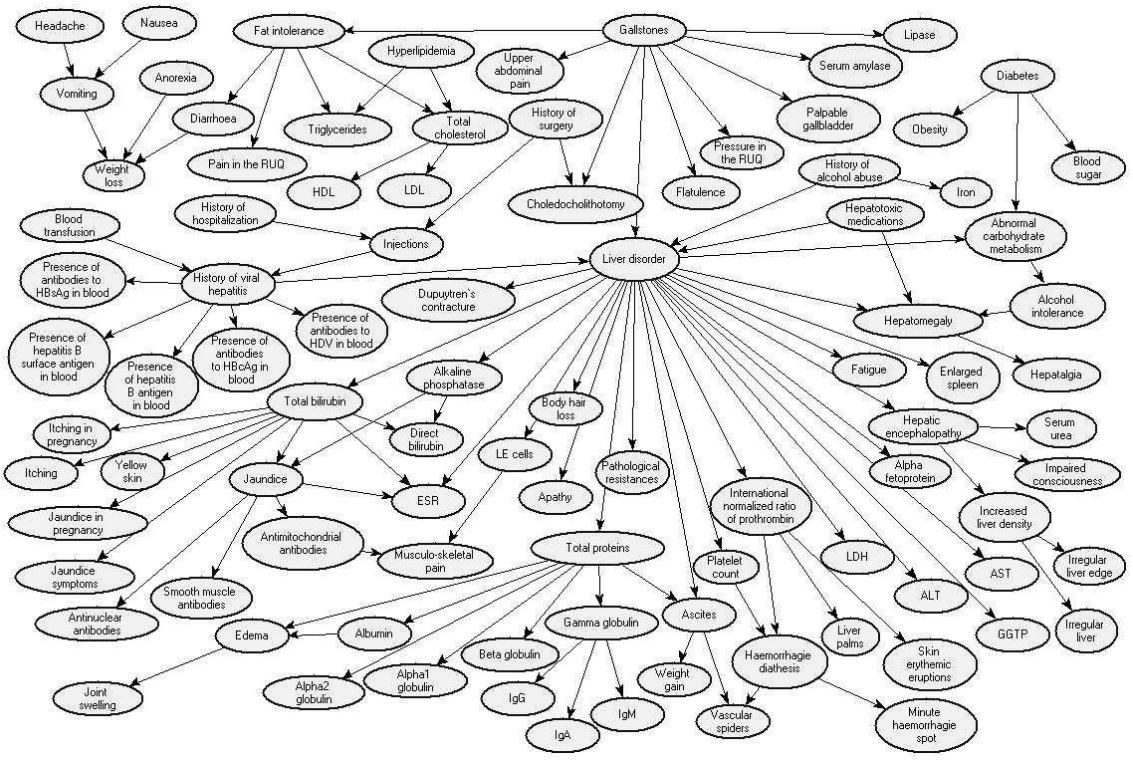
***Credit Score*:** Existem diversas empresas de crédito que utilizam algoritmos feitos a partir de Redes Bayesianas para estimar se determinada pessoa será boa

pagadora ou não, por exemplo [29]. A *BayesCredit* é um sistema criado pela empresa Dinamarquesa *Nykredit,* especializada em financiamento de imóveis [29].

Predição de Reprovação de Alunos da modalidade EAD: Foi apresentado no III Congresso Brasileiro 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 [35]. Alguns dos atributos usados na rede foram: Situação acadêmica, Interações por Semana e o Fator de Empenho.

Diagnóstico de doenças do fígado: Com o objetivo de diagnosticar doenças hepáticas, o modelo bayesiano proposto por [3] baseia-se em um banco de dados clínico composto por 16 doenças do fígado e 93 características (sintomas importantes, sinais e fatores de risco). O cálculo é feito a partir do subconjunto de pelo menos uma dessas 93 caraterísticas. Por fim é encontrada a distribuição de probabilidade posterior ao longo dos possíveis 16 distúrbios hepáticos. A figura 3.3 ilustra a estrutura do modelo para o diagnóstico de distúrbios do fígado. Tal modelo foi pensado para ser aplicado no treinamento médico e na prática clínica. Em estudos posteriores estima-se a utilização de opiniões de especialistas para extrair os parâmetros numéricos da rede.

Figura 3.3: Estrutura do modelo para o diagnóstico do fígado [3]



Monitoramento de pacientes em terapia intensiva: O método para a aprendizagem de estruturas de Redes Bayesianas através da aplicação dos algoritmos de Markov Chain e Monte Carlo proposto por [4] teve o objetivo de realizar o monitoramento em pacientes em terapia intensiva. Foram utilizados dois modelos de redes. O primeiro foi criado por especialistas do hospital universitário da Universidade Federal de Santa Catarina cujas variáveis foram os dados antropométricos de atividade física, pressão arterial e avaliação do estado nutricional dos pacientes (figura 3.4). O segundo modelo utilizado baseou-se na rede *Alarm*, idealizada por [36] para monitorar pacientes em terapia intensiva disponível na figura 3.5.

Figura 3.4: Estrutura da rede Paciente HU criada pelo especialista [4].

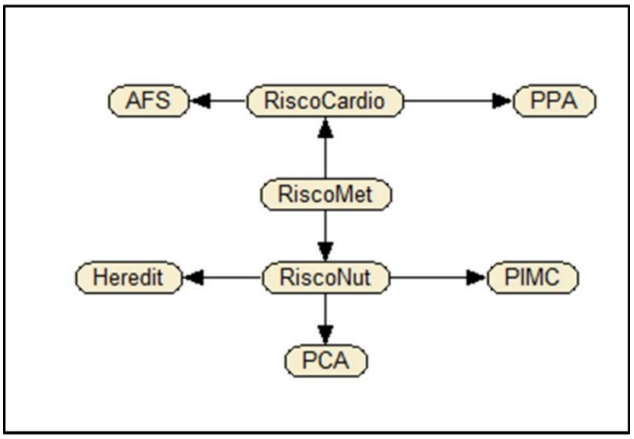
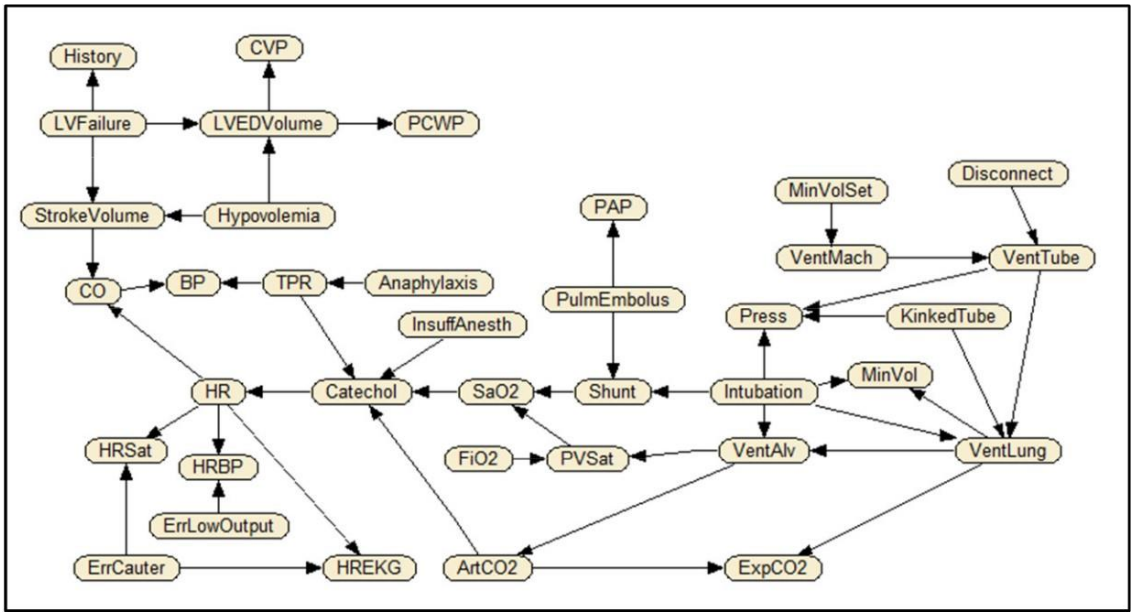


Figura 3.5: Estrutura da rede *Alarm* [4].



Dessa forma, os algoritmos de inferência de Rede Bayesiana conferem um método popular para modelar domínios complexos. Eles fornecem uma estrutura poderosa e matematicamente coerente para a análise de problemas que envolvam a captura de dados lineares e não lineares, padrões combinatórios, estocásticos, entre outros tipos de relações entre variáveis em inúmeros níveis de organização, como em um ecossistema, por exemplo. A arquitetura modular de uma rede bayesiana permite o desenvolvimento de modelos iterativos.

4. MODELOS DE MATURIDADE, GERENCIA DE TESTES E QUALIDADE

Para que uma organização seja conduzida à maturidade no âmbito dos processos de desenvolvimento de *software*, são necessários muito mais do que modelos e/ou normas capazes de conferir e garantir robustez as etapas para produção do *software.* Os modelos são responsáveis por estabelecer as práticas necessárias para a gestão dos processos de *software*, contudo é necessária uma mudança de cultura e comprometimento da equipe na empresa ou no departamento de desenvolvimento.

A abordagem utilizada nesta obra valoriza as pessoas, respeita o investimento no projeto, utilizando os métodos ágeis, e está em conformidade com as premissas do CMMI, da ISO 15504 e do MPS.BR. Para melhor entendimento do que se deseja alcançar, serão apresentados os principais modelos do mercado.

4.1 CMMI

O SEI (*Software Engineering Institute*), instituto integrado à universidade norte-americana Carnegie Mellon elaborou o CMMI (*Capability Maturity Model Integration*), o modelo integrado de maturidade em capacitação que encontra-se na versão 1.3. As orientações geradas pelo CMMI têm o objetivo de incentivar a melhoria de processos em organizações de qualquer estrutura [37].

Tal modelo parte do princípio de que muitas organizações especializadas no desenvolvimento de *software*, por exemplo, não têm condições de seguir todos os processos do ciclo de desenvolvimento.

Por isso o CMMI sugere que um maior nível de maturidade pode ser alcançado quando as organizações adotam as melhores práticas para a melhoria de seus processos/produtos.

Tal maturidade é conquistada de maneira gradativa e permite que haja uma

maior qualidade na produção de *software*, e com isso uma menor chance gerar de erros.

O CMMI é subdividido em três modelos de referência de acordo com áreas de interesse [38]: O CMMI para desenvolvimento (*CMMI-DEV*), tem como objetivo o desenvolvimento de produtos e serviços.

Ele é aplicável tanto para serviços simples quanto para o desenvolvimento de sistemas robustos e complexos; O CMMI para aquisição (*CMMI-ACQ*), tem como foco as atividades para iniciar e gerenciar a aquisição e terceirização de produtos e serviços; O CMMI para serviços (*CMMI-SVC*), é um modelo que viabiliza um conjunto integrado de medidas para organizações prestadoras de serviços. Assim esse modelo busca o fornecimento de serviços de qualidade para clientes e usuários finais.

Para selecionar o modelo adequado devem ser considerados o foco principal da empresa e o ciclo de vida em que seus processos/produtos estão inseridos. O CCMI é flexível a ponto de suportar dois tipos diferentes representações, uma em estágios (absorvida do CMM) baseada em níveis de maturidade e uma outra contínua baseada em níveis de capacidade, compatível à norma ISO/IEC 15504-2 [39, 40]. Cada organização escolhe a representação de acordo com os resultados esperados ou com a qual está mais familiarizada [37].

Níveis de Capacidade

Nesse modelo é possível medir a maturidade de uma empresa com base na capacidade do processo. Há a possibilidade de serem escolhidas as práticas e a ordem em que devem ser seguidas de maneira que melhor atendam aos objetivos de negócios da organização promovendo uma redução de suas áreas de risco. Assim, as áreas de processos são avaliadas individualmente [37, 40]. O modelo ainda permite a comparação entre duas organizações levando também em consideração cada área de processo ou suas amostras equivalentes.

Nesse aspecto o CMMI está voltado também para a avaliação de processo de *software*. A representação contínua do CMMI possui quatro níveis: Nível 0: Incompleto: em andamento. Nível 1: Executado: em andamento. Nível 2: Gerenciado: em andamento. Nível 3: Definido: em andamento.

Níveis de Maturidade

Os níveis de maturidade promovem melhorias no desempenho geral das organizações a partir de práticas básicas de gerenciamento. Isso ocorre por meio de passos estipulados e divididos entre níveis sucessivos, cada um atuando como base do próximo [37]. Para atingir um nível de maturidade superior, as organizações devem alcançar as melhorias do nível anterior.

Além disso, é possível comparar o nível de maturidade entre organizações que adotam esse tipo de representação. Existem cinco níveis de maturidade, sendo que o quinto e o primeiro níveis indicam o maior e o menor grau de maturidade. Os níveis de maturidade estão listados na tabela 2.3.

Tabela 2.3: Níveis de Maturidade do CMMI [37].

1 Inicial	Processo imprevisível
2 Gerenciado	Processos básicos de gerenciamento
3 Definido	Padronização de processos
4 Quantitativamente Gerenciado	Quantitativamente controlado
5 Em otimização	Melhoria contínua do processo

4.2 ISO/IEC 15504

A ISO/IEC 15504 conhecida também como SPICE (*Software Process Improvement and Capability Determination*) foi criada em 1993 e constitui um padrão internacional utilizado na avaliação de processos de *software*. Ela define diversos processos de engenharia de *software*, como a análise de requisitos. Além disso a norma fornece uma escala para a medição da capacidade dos processos, de modo que um melhor desempenho do projeto está relacionado com uma maior capacidade do processo [41, 42].

O SPICE é dividido em cinco partes de caráter informativo com exceção da segunda parte que é explicitamente normativa [43, 44]: Parte 1 - Conceitos e vocabulário. Parte 2 - Execução de uma avaliação. Assim como o CCMI, a ISO 15504-7 apresenta níveis de capacidade que individualmente fornecem um conjunto de processos capazes de caracterizar diversos comportamentos organizacionais. No SPICE, os níveis de capacidade são identificados através da escala de atributos do processo [44, 45]: Nível 0: Incompleto - Sem atributos. Nível 1: Executado - Execução

do processo. Nível 2: Gerenciado - Gerência da execução, gerência do produto. Nível 3: Estabelecido - Definição de processo, recursos de processo. Nível 4: Previsível - Medição de controle e controle de processo. Nível 5: Otimizando - Mudança de processo e melhoria contínua.

Parte 3 - Instrução para a execução de uma avaliação. Parte 4 - Instrução para utilização dos resultados da avaliação. Parte 5 - Exemplo de modelo de avaliação de processo baseado na ISO/IEC 12207.

A partir de um perfil criado com o objetivo de realizar a melhoria de processos, uma empresa gera o plano dessas melhorias contendo as qualidades desejáveis e indesejáveis, assim como os riscos inerentes aos processos. Em contrapartida, uma organização pode criar o perfil de capacidade de fornecedor em potencial, como forma de avaliação do mesmo [43, 46, 41].

4.3 MPS.BR

Com o objetivo de levar melhorias à capacidade de desenvolvimento de *software* nas empresas brasileiras, o MPS-BR (Melhoria de Processos do *Software* Brasileiro), foi criado em 2003 pela Associação para Promoção da Excelência do *Software* Brasileiro - Softex. O MPS-BR é um modelo de qualidade de processos baseado na definição de processos, propósitos e resultados da ISO/IEC 12207, na definição de processos e requisitos de avaliação da ISO/IEC 15504 e na complementação de processos do CMMI. Assim o MPS-BR treina e certifica pessoas, micro, pequenas e médias empresas nos processos de desenvolvimento de *software* [47].

O MPS-BR possui sete níveis de maturidade (do A ao G) relativos às áreas de processos baseadas nos níveis estagiados do CMMI (Figura 4.1) [5, 6].

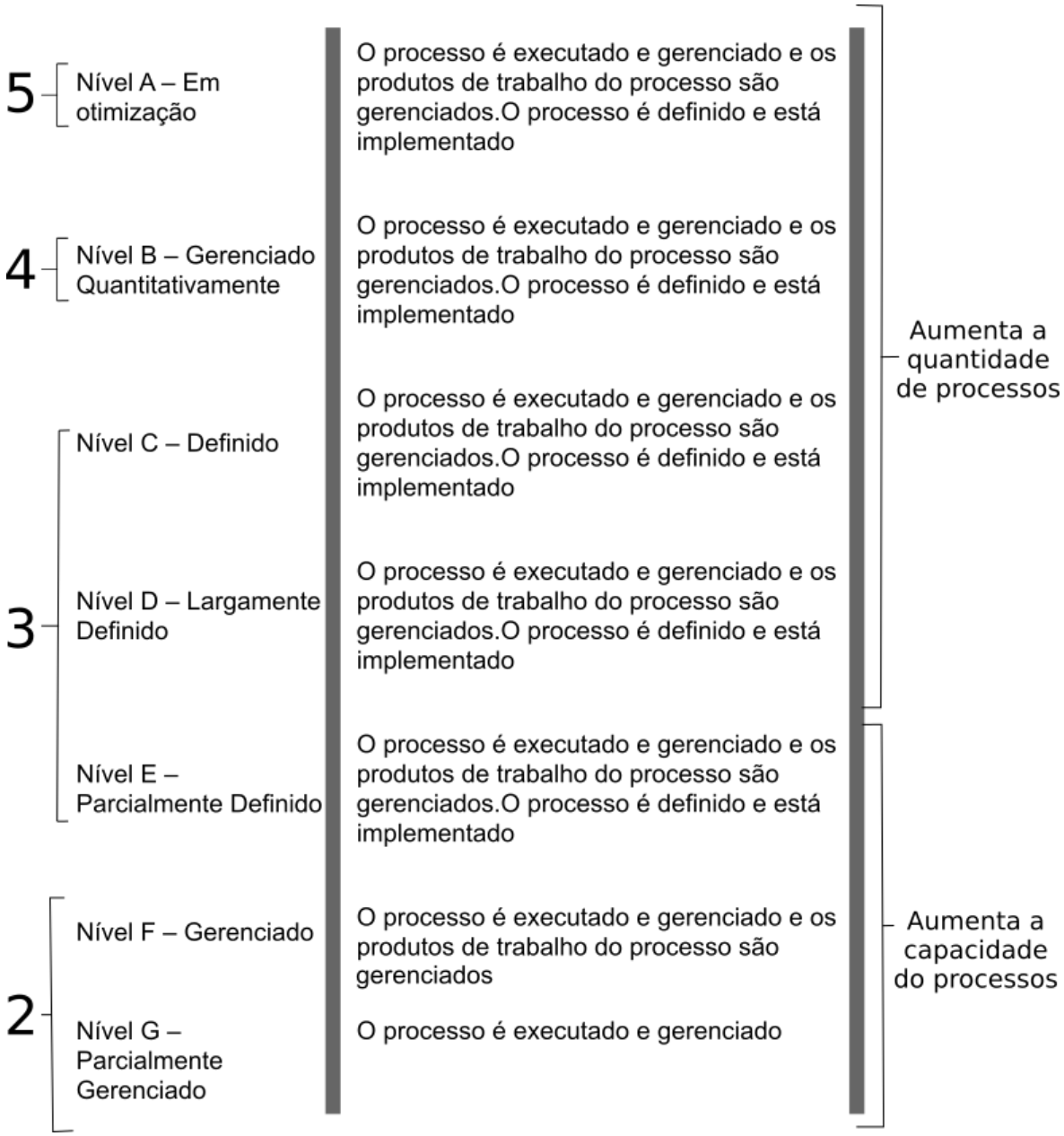
Enquanto dos níveis A ao D há uma maior quantidade de processos, dos níveis E ao G a capacidade dos processos é aumentada. Portanto os níveis de maturidade do MPS-BR constituem uma combinação entre processos e sua capacidade [47, 5].

4.4 AADSP

A abordagem AADSP - *Adaptive Approach for Deployment of Software Process* - Abordagem Adaptativa para Implantação de Processo de *software* proposta pelo Labrasoft - Laboratório de Desenvolvimento de *Software* - em 2014 e financiado pelo edital de fortalecimento de pesquisa da Pró-Reitoria de Pesquisa, Pós-Graduação e

Inovação (PRPGI) do IFBA tem como alicerce práticas inovadoras em conformidade com o modelo MPS-BR da SOFTEX, práticas dos métodos ágeis, demandas dos profissionais da área de análise e desenvolvimento de sistemas, interesse das empresas ou departamento cliente no acompanhamento dos projetos de *software*.

Figura 4.1: Níveis de maturidade do MPS-BR correlacionados aos níveis em estágios do CMMI [5, 6]



O objetivo da abordagem AADSP é promover a qualidade de *software* por meio do equilíbrio na utilização de métodos ágeis e de melhoria de processos de *software.* O AADSP tem cinco pilares: 1- Bem-estar e realização das pessoas envolvidas: O

grande diferencial nos projetos bem-sucedidos é o comprometimento, colaboração e auto-organização dos profissionais envolvidos (desenvolvedores internos, líderes e gerentes de desenvolvimento e sistemas, *freelancers*, *stakeholders*, cliente ou financiadores do projeto de *software*); 2- Transparência e melhoria da comunicação com o cliente ou consumidor: O entendimento e participação do cliente no projeto precisa ser; 3- Facilitado com linguagens comuns para que este entenda o andamento do projeto e o que está sendo feito; 4- A empresa desenvolvedora pode pré-avaliar seus projetos com base em artefatos e uso da ferramenta em conformidade com a AADSP, gerando *dataset* para pesquisa e medição; 5- Melhoria contínua do Retorno por Investimento (ROI) para todos os envolvidos através da reutilização de artefatos, da melhor utilização do tempo e redução de retrabalho.

4.4.1 GERÊNCIAS DO AADSP

Com base nas gerências propostas pelo guia [48], o AADSP catalogou seis gerências que tem como principal objetivo definir um conjunto de artefatos que possibilite a qualidade dos *softwares*.

Gerência de Requisitos e Modelagem

Gerenciar os requisitos e componentes do projeto, controlar a evolução e identificar inconsistências entre os requisitos, os planos do projeto e os produtos de trabalho do projeto além de proporcionar uma visão comum entre a alta gerência e os *stakeholders* [48]; 1. Documento de Requisitos; 2. Documento de Solicitação de Mudanças; 3. Análise de Impacto em modificações em Requisitos; 4. Histórico e Versionamento dos Requisitos; 5. Sinalizar Inconsistências e Correções; 6. Matriz de Rastreabilidade entre Requisitos; 7. Matriz de Rastreabilidade entre Requisitos e Artefatos.

Gerência de Projetos

Estabelecer e manter planos de trabalhos que definam atividades, cronogramas, recursos e os responsáveis pelo projeto [48]: 1. Termo de Abertura do Projeto: Documento de autorização formal de início do projeto e que possibilita ao gerente do projeto a autoridade para aplicar os recursos necessários para a execução do projeto; 2. Plano do Projeto: Documento que possui todas as ferramentas e metodologias a serem utilizadas para a execução do projeto; 3. Ata de Reuniões.

Gerência de Configuração e Mudanças

O escopo do projeto e os seus requisitos podem ser modificados a qualquer momento durante o ciclo de vida do *software*. Assim, requisitos adicionais podem ser incorporados no projeto, os requisitos podem ser removidos do projeto e/ou as mudanças podem ser feitas para os requisitos existentes [48]; 1. Documentos comprovatórios de solicitações de mudanças; 2. Avaliação dos impactos positivos e negativos que uma modificação pode gerar; 3. A empresa trabalha com repositórios distribuídos para gerenciamento de configuração e mudança; 4. A empresa utiliza Sistemas de Controle de Versão para melhor gerenciar seus documentos código-fonte.

Gerência de Colaboradores e *Stakeholders*

Organizar, gerenciar, delegar responsabilidades para membros envolvidos na construção do *software* sejam eles desenvolvedores ou *stakeholders* que conhecem as regras do negócio [48]. 1. Planejamento das necessidades estratégicas da organização e dos projetos para identificar recursos, conhecimentos e habilidades requeridos e, de acordo com a necessidade, planejar como desenvolvê-los ou contratá-los; 2. Para todo e qualquer projeto o conjunto de pré-requisitos para o desenvolvimento do mesmo deve ser elencado para que treinamentos efetivos e objetivos possam ser aplicados; 3. Os artefatos com regras de negócios devem ser amplamente difundidos para todos os indivíduos que possuírem a devida permissão de acesso de acordo com seu cargo e grau de interesse; 4. Dar um contínuo retorno sobre o progresso das atividades para a equipe de desenvolvimento bem como aos *stakeholders* do projeto.

Gerência de Testes

Desenvolver planos e estratégias para implementação de teste e inspeção de *software* objetivando a melhoria na qualidade e integridade dos dados apresentados nos produtos finais [48]; 1. Planos de Testes para o Projeto. Esses planos de teste podem estar vinculados ao conjunto de funcionalidades e requisitos de um *software*; 2. Lista de ocorrências de erros e não conformidade; 3. Lista de ações corretivas; 4.

Controle de inspeção e qualidade; 5. Termo de formação da equipe de qualidade; 6. Documento de homologação do teste de *software*; 7. Glossário de erros do projeto.

Gerência de Reuso

Fornecer maiores artefatos de usabilidade produzidos pelos projetos por mapeamento e documentação de componentes e outros ativos reutilizáveis de um *software* [48]; 1. Definir a padronização do código-fonte elaborado para cada projeto de *software*; 2. Disponibilizar o conjunto de artefatos reutilizáveis em uma base de fácil acesso aos colaboradores e manter um *backup* da mesma; 3. Definição clara de ativos reutilizáveis ou que possam ser reutilizados.

A abordagem AADSP é voltada para a entrega de artefatos, e deste modo, os processos, qualidade e controle de mudanças são comprovados a partir da existência e comprovação dos mesmos em projetos de *software* que implementem esta metodologia. Diferente do [49], essa abordagem é mais flexível e pode proporcionar melhorias aos produtos entregues por micro e pequenas empresas e por *freelancers*.

Para compreensão da conformidade entre o AADSP e os modelos de maturidade apresentados, são apresentados, no próximo capítulo, os artefatos da Gerência de Testes. No nível F da Melhoria do Processo de *Software* Brasileiro (MR-MPS), o interesse para este livro é a Garantia da Qualidade e Medição. Também fazem parte deste nível a composição dos artefatos através da Aquisição, Gerência de Configuração, Gerência de Problemas e Gerência de Portfólio de Operação de Serviços bem como os processos de maturidade do nível G [47].

CAPÍTULO 02

ARTEFATOS

5. ARTEFATOS E DOCUMENTAÇÃO

No presente capítulo serão considerados os itens relacionados à Gerência de Qualidade. Sua finalidade é a equivalência entre a execução dos processos e os produtos de trabalho com os planos e recursos previamente determinados. Com o objetivo de alcançar essa garantia, sete artefatos devem ser implementados. Tanto os artefatos como os itens que os compõem possuem um grau de importância, já que nem todas as organizações são capazes de implementá-los em sua totalidade, principalmente quando estão em processo de amadurecimento.

5.1 PLANO DE EXECUÇÃO DE TESTES DO PROJETO (ESSENCIAL)

Este plano consiste num catálogo que deverá conter os objetivos ou resultados esperados para um determinado artefato do projeto de *software* [50]. Ele integra as tarefas de teste no projeto e serve como um mecanismo para estabelecer a comunicação entre os *stakeholders*. O plano de testes pode ser considerado como um guia para o controle das tarefas de teste e sua execução. Os itens de um plano de teste compreendem: Introdução; Requisitos que devem ser testados; Ferramentas e estratégias de teste; Infraestrutura; Equipe; Cronograma de atividades; Documentação complementar.

5.2 LISTA DE OCORRÊNCIA DE ERROS E NÃO CONFORMIDADES (IMPORTANTE)

As diferenças encontradas entre o esperado e o realizado no momento da avaliação de garantia da qualidade do produto e do processo são consideradas como não-conformidades ou erros relativos à melhoria dos processos executados. Após detectadas tais diferenças devem ser listadas em um documento onde serão encontrados todos os erros relativos ao processo de desenvolvimento do *software.*

Por fim, essa lista deverá ser fornecida a todas as partes interessadas para que em um momento adequado os erros possam ser sanados.

5.3 LISTA DE AÇÕES CORRETIVAS (IMPORTANTE)

Após a criação e divulgação dos documentos contendo os erros, uma lista deverá ser criada com as devidas ações necessárias para a correção deles. As ações corretivas devem ser determinadas e acompanhadas até que a não-conformidade seja resolvida. Assim, para cada não-conformidade deverá haver um registro na lista de ações corretivas, as quais podem tanto em conjunto com outras ações corretivas, quanto sozinhas resolver um ou mais problemas descritos lista de erros.

5.4 CONTROLE DE INSPEÇÃO E QUALIDADE (ESSENCIAL)

Neste documento os defeitos dos artefatos de *software* são rigorosamente inspecionados e definidos desde o momento de sua detecção através da revisão de um artefato, até o direcionamento para que o autor deste artefato os corrija. No registro deverá constar uma avaliação final indicando a necessidade, ou não de uma nova inspeção.

5.5 TERMO DE FORMAÇÃO DA EQUIPE DE QUALIDADE (DESEJÁVEL)

Durante o processo de desenvolvimento do *software* deve ser definida a equipe responsável pela qualidade do produto que está sendo desenvolvido. O AADSP não restringe que os membros da equipe façam parte da própria equipe de desenvolvimento do produto, ressalta-se que, as decisões tomadas devem ser pela equipe evitando inconsistências aos resultados esperados [50].

5.6 DOCUMENTO DE HOMOLOGAÇÃO DO TESTE DE *SOFTWARE* (IMPORTANTE)

O documento de homologação do teste de *software* é um artefato preparado pela equipe de desenvolvimento de *software* em conjunto com a equipe de gerência de contratos da empresa ou do seu departamento de tecnologia. Tal documento deve ser assinado pela empresa ou pelo departamento/cliente a fim de validar os requisitos funcionais e não funcionais definidos no documento de requisitos.

Através desse documento atesta-se que o produto desenvolvido no projeto de *software* ampara os critérios estabelecidos com os clientes. O documento de homologação do teste de *software* formaliza a responsabilidade e a adoção do *software* ou produto e só é iniciado após a conclusão de todos os testes.

Assim, todos os erros encontrados devem ser corrigidos antes do final da homologação ou aceitos. Com a homologação objetiva-se: Comprovar que o *software* age da forma esperada pelo cliente; Encontrar e corrigir os erros; Adquirir a aprovação do produto de *software* por parte do cliente; Obter sugestões para o caso de versionamento da aplicação.

O documento de homologação do teste de *software* formaliza a responsabilidade e a adoção do *software* ou produto.

Os itens que compõem o documento de homologação são: Requisitos funcionais de cada módulo; Requisitos não funcionais de cada módulo; Responsáveis da empresa ou departamento cliente; Data da homologação; Módulos; Sugestões de melhorias futuras.

5.7 GLOSSÁRIO DE ERROS DO PROJETO (IMPORTANTE)

O glossário de erros do projeto é o artefato que deverá conter os erros ocorridos no projeto e as soluções utilizadas para resolvê-los evitando deste modo redundâncias de resoluções de erros encontrados [50]. A lista contida neste glossário corresponde a: Erros de análise; Erros de projeto; Erro de modelagem de dados; Erro na codificação; Erro na implantação; Erros na construção ou no uso em qualquer um dos artefatos citados nas subseções anteriores.

CAPÍTULO 03
CENÁRIOS DE AVALIAÇÃO

6. CENÁRIOS DE AVALIAÇÃO

Nesse capítulo serão elencados o conjunto de critérios para a avaliação dos artefatos requeridos pela abordagem AADSP para a gerência de testes e a validação da *Rede Bayesiana* construída.

6.1 AVALIAÇÃO DOS ARTEFATOS

Para o processo de avaliação e auditoria dos artefatos a abordagem AADSP propõe o uso do paradigma GQM *(Goal/Questions/Metrics)* em conjunto com a escala *Likert.* Amplamente utilizada na Engenharia de *Software* o GQM baseia-se na definição de uma Meta (*Goal*) e um conjunto de Questões *(Questions)* são estabelecidas para auxiliar na compreensão da Meta, além disso, um conjunto de métricas (*Metrics*) são estabelecidas para avaliar essas questões.

Para auxiliar no processo de auditoria e validação dos artefatos, a AADSP sugere a substituição das Métricas do GQM pela avaliação com a Escala *Likert* [51] que proporciona uma avaliação quantitativa para as questões. Todas as questões avaliativas devem ser respondidas seguindo o formato tradicional da Escala *Likert:* 1. Discordo totalmente (1); 2. Discordo parcialmente (2); 3. Indiferente (3); 4. Concordo parcialmente (4); 5. Concordo totalmente (5).

Cada artefato possui um conjunto de **Questões Avaliativas** (QA) que serão respondidas conforme a Escala Likert acima representada. A pontuação que o artefato receberá será composto pelo somatório de todas as QAs divido pela quantidade de QAs do artefato em questão.

6.2 AVALIAÇÃO DA REDE BAYESIANA PROPOSTA

Para realizar o processo de avaliação do AADSP - Gerência de Testes, o primeiro passo é calibrar/treinar a Rede Bayesiana. Para nosso exemplo, serão apresentados três cenários sintéticos, com dois requisitos cada.

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.

6.2.1 VALIDAÇÃO USANDO DADOS SINTÉTICOS

Cenário 01:

Tabela	**Requisito A**	**Requisito B**
Participação de desenvolvedores	Confiável (5,0)	Muito confiável (8,3)
Cobertura de testes anteriores	Baixa (30 %)	Média (45 %)
Impacto das alterações realizadas	Alto (80 %)	Médio (50 %)
Nível de importância do requisito para o *stakeholder*	Alto (9.0)	Médio (5.0)
Nível de importância do requisito para o projeto	Alto (10.0)	Média (6.0)
Nível de importância do *stakeholder* para o projeto	Alto (8.0)	Alta (8.0)
Valor do projeto	Médio (R$ 45.000,00)	Baixo (R$ 15.000,00)
Prazo de entrega	Curto (2 meses)	Curto (2 meses e meio)

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 saíram 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 *stakeholders* 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.

Cenário 02:

Tabela	**Requisito C**	**Requisito D**
Participação de desenvolvedores	Confiável (6.0)	Confiável (5.0)
Cobertura de testes anteriores	Média (55 %)	Média (45 %)
Impacto das alterações realizadas	Médio (65 %)	Baixo (35 %)
Nível de importância do requisito para o *stakeholder*	Médio (6.0)	Médio (5.0)
Nível de importância do requisito para o projeto	Médio (6.0)	Baixo (2.0)
Nível de importância do *stakeholder* para o projeto	Médio (5.0)	Alto (8.0)
Valor do projeto	Médio (R$ 50.000,00)	Médio (R$ 65.000,00)
Prazo de entrega	Curto (3 meses e meio)	Médio (4 meses)

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, só 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 média, porque a sua nota de importância 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 entrega é curto, que lhe dá 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.

Cenário 03:

Tabela	**Requisito E**	**Requisito F**
Participação de desenvolvedores	Muito Confiável (9.0)	Muito Confiável (8.0)
Cobertura de testes anteriores	Alta (86 %)	Alta (75 %)
Impacto das alterações realizadas	Médio (65 %)	Baixo (35 %)
Nível de importância do requisito para o *stakeholder*	Médio (5.0)	Baixo (3.0)
Nível de importância do requisito para o projeto	Médio (5.0)	Baixo (2.0)
Nível de importância do *stakeholder* para o projeto	Baixo (2.0)	Alto (8.0)
Valor do projeto	Médio (R$ 45.000,00)	Médio (R$ 55.000,00)
Prazo de entrega	Longo (7 meses e meio)	Médio (5 meses e meio)

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.

6.2.2 VALIDAÇÃO USANDO DADOS REAIS

Utilizando os dados reais de três projetos de uma empresa de desenvolvimento de *software*, a predição dos requisitos foi feita em três *sprints* separadas. Dessas três *sprints*, foram retirados os cinco requisitos mais prioritários dentro do projeto.

Sprint 1: A primeira *Sprint* teve como entrada trinta requisitos dos três projetos diferentes e, da saída, foram extraídos os seguintes requisitos:

	Atualizar Visitante	**[SDV024]**	**Consulta compras beneficiário**	**Controle de Acesso**	**Cadastro beneficiários lote**
Nível de impacto de alterações	Média	Alta	Alta	Alta	Alta
Nível dos desenvolvedores	Pouco confiável	Confiável	Confiável	Muito Confiável	Muito Confiável
Porcentagem de cobertura dos testes anteriores	Mínima	Mínima	Mínima	Mínima	Cobertura Mínima
Nível de importância do *stakeholder* para o requisito	Alta	Alta	Alta	Alta	Alta
Nota de prioridade do requisito	Alta	Alta	Alta	Alta	Alta
Nível de importância do *stakeholder* para o projeto	Média	Média	Alta	Média	Alta
Custo do projeto	Alto	Baixo	Médio	Alto	Médio
Prazo de entrega	Médio	Curto	Longo	Médio	Longo
PRIORIDADE ALTA	74 %	72 %	72 %	72 %	69
PRIORIDADE MÉDIA	19 %	18 %	18 %	21 %	21
PRIORIDADE BAIXA	7 %	9 %	9 %	8 %	11

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:

	Login	**Consultar SCI Licitação**	**Cadastrar Conselheiro**	**Atualizar Conselheiro**	**Editar convenio**
Nível de impacto de alterações	Alta	Alta	Média	Média	Média
Nível dos desenvolvedores	Confiável	Pouco Confiável	Confiável	Confiável	Muito Confiável
Porcentagem de cobertura dos testes anteriores	Alta	Média	Média	Média	Média
Nível de importância do *stakeholder* para o requisito	Alta	Baixa	Alta	Alta	Alta
Nota de prioridade do requisito	Alta	Média	Alta	Alta	Alta
Nível de importância do *stakeholder* para o projeto	Alta	Média	Média	Média	Alta
Custo do projeto	Médio	Alto	Alta	Alta	Médio
Prazo de entrega	Longo	Médio	Média	Média	Longo
PRIORIDADE ALTA	60 %	55 %	55 %	55 %	54 %
PRIORIDADE MÉDIA	26 %	30 %	34 %	34 %	31 %
PRIORIDADE BAIXA	14 %	15 %	11 %	11 %	15 %

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:

	Consultar Convenio	**Movimentos por comerciante**	**Cadastrar Endereço ADM**	**[SDV004]**	**Bloquear Cartão**
Nível de impacto de alterações	Alta	Média	Alta	Alta	Baixa
Nível dos desenvolvedores	Confiável	Confiável	Muito Confiável	Muito Confiável	Muito Confiável
Porcentagem de cobertura dos testes anteriores	Alta	Alta	Média	Média	Média
Nível de importância do *stakeholder* para o requisito	Alta	Alta	Alta	Alta	Alta
Nota de prioridade do requisito	Média	Alta	Baixa	Baixa	Alta
Nível de importância do *stakeholder* para o projeto	Alta	Alta	Alta	Média	Alta
Custo do projeto	Médio	Média	Médio	Baixo	Médio
Prazo de entrega	Longo	Longo	Longo	Curto	Longo
PRIORIDADE ALTA	50 %	49 %	48 %	48 %	47 %
PRIORIDADE MÉDIA	30 %	34 %	29 %	29 %	35 %
PRIORIDADE BAIXA	20 %	17 %	22 %	22 %	18 %

CAPÍTULO 04

FERRAMENTAS

7. FERRAMENTAS

7.1 AVALIAÇÃO DAS FERRAMENTAS

Nessa seção as principais ferramentas de gerenciamento de caso de teste disponíveis são apresentadas:

T-AADSP Test[8]**:** Ferramenta desenvolvida pelo grupo de pesquisa *Labrasoft* - Laboratório de Desenvolvimento de *software* - do IFBA, a aplicação visa auxiliar no gerenciamento de testes de um projeto de *software* com base na abordagem AADSP e também com as principais funcionalidades que um gerenciador de casos de teste precisa.

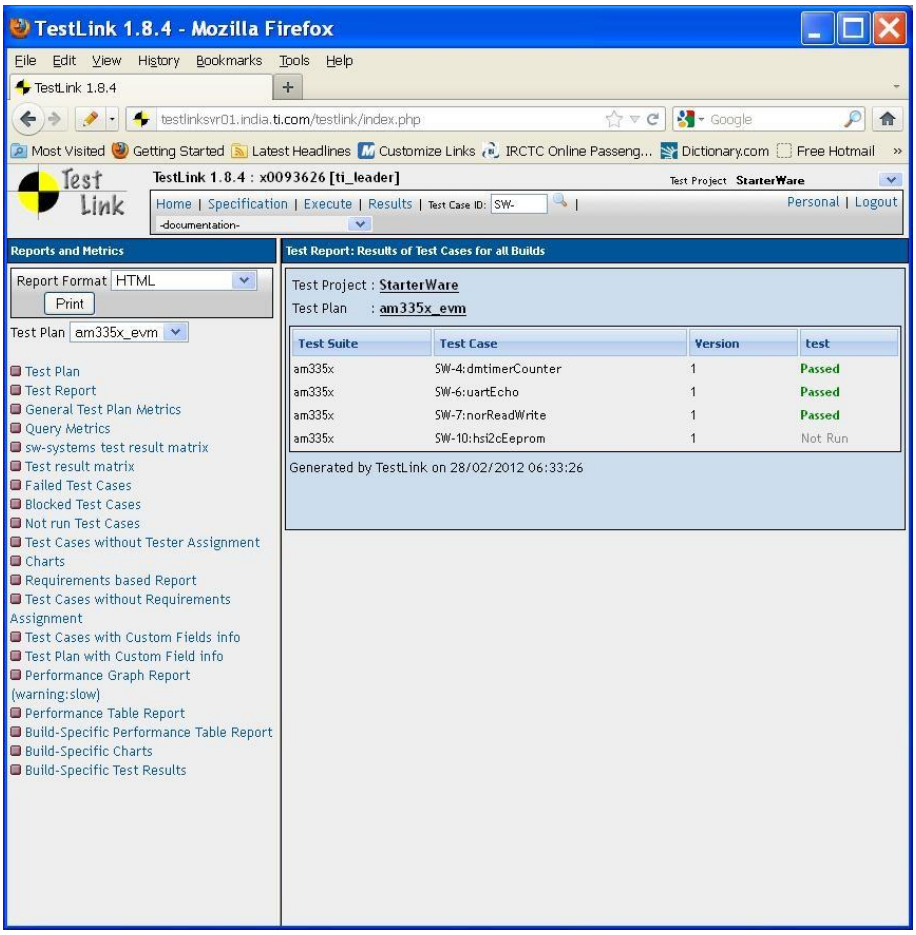
Tem como principal diferencial a adoção de uma Rede Bayesiana para realizar a predição dos casos de testes de determinada funcionalidade, sendo assim escolhido os casos de testes mais essenciais de serem testados naquele momento do teste.

TestLink[2]**:** É uma ferramenta de gerenciamento de testes de *software* de código aberto que permite que a equipe, tanto de desenvolvimento quanto de teste trabalhem de forma sincronizadas, a sua *interface* é *web* e permite a divisão das pessoas do projeto em níveis de acessos (líder, testador, desenvolvedor, etc). A figura 7.1 apresenta a interface do TestLink no momento em que o teste de um projeto é executado.

Sua matriz de resultados do teste fornece relatórios e métricas. Entretanto, esta não é considerada uma ferramenta completa, uma vez que a mesma não possui um gerenciador de erros. Para tanto, é necessário integrá-la com rastreadores de erros conhecidos como *bug trakcers*, como o Mantis (figura 7.2).

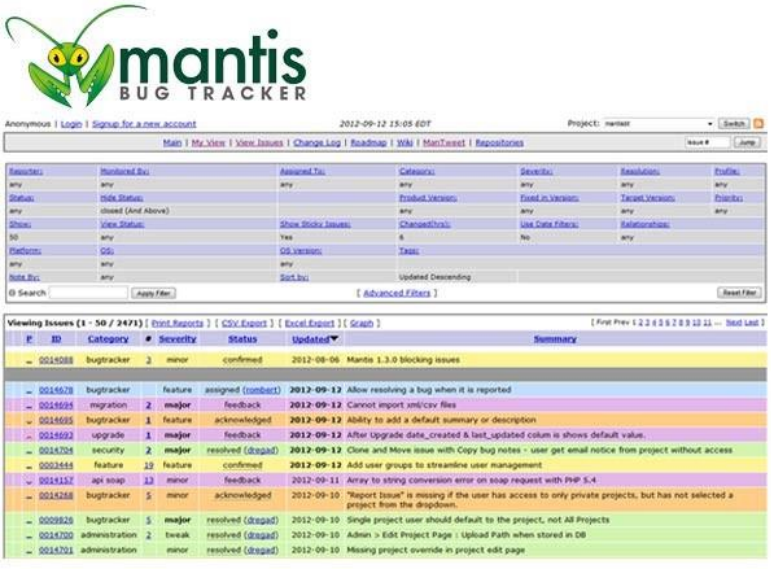
[8] http://www.labrasoft.ifba.edu.br/canal-aadsp/aadsp/t-aadsp-test [2]http://testlink.org/

Figura 7.1: Resultado da execução de um teste na ferramenta TestLink.



Fonte: https://processors.wiki.ti.com/index.php/StarterWare_Test_Framework

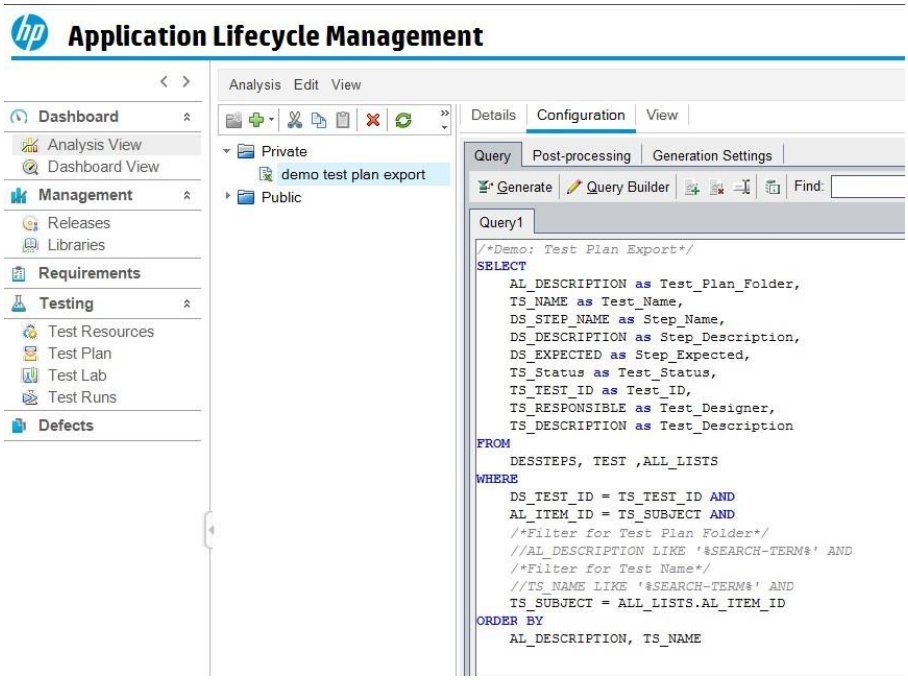
Figura 7.2: Interface do Mantis, um rastreador de erros em teste de projetos.



Fonte: http://www.clubedatecnologia.com.br/2017/07/

ALM[9]**:** O ALM, figura 7.3 é uma suíte proprietária completa de gerenciamento do ciclo de testes que permite acompanhar a etapa da execução dos testes de ponta a ponta (desde o começo dos testes até a criação dos *bugs*). Sua interface permite a monitoração detalhada dos status dos testes em tempo real, além de calcular automaticamente a cobertura dos testes executados para a funcionalidade e a criação de registro de *bugs*. Os *bugs* podem ser facilmente vinculados aos casos de testes que os encontraram.

Figura 7.3: Interface do ALM.

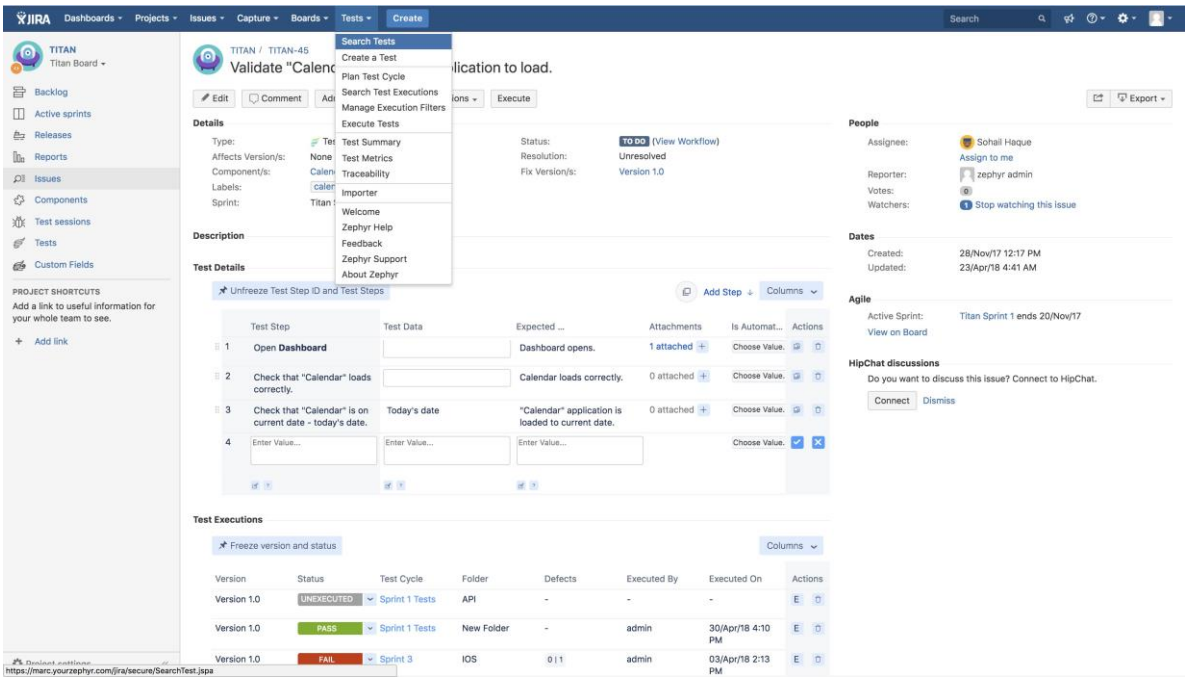


Fonte: https://blogs.sap.com/2018/12/23/transfer-test-cases-from-micro-focus-alm-tosap-solution-manager-test-steps/

Zephyr for Jira[10]**:** Ferramenta de gerenciamento de caso de testes proprietária que foi desenvolvida para ser um *plugin* para a ferramenta Jira. O Zephyr permite a criação e edição de casos de testes dentro do Jira, e a execução desses casos de testes quando for necessário. Ele também fornece suporte a métricas de testes detalhadas e o vínculo de um ou mais casos de testes a *tickets* do Jira (figura 7.4).

[9] https://www.microfocus.com/pt-br/products/application-lifecyclemanagement/overview
[10] https://marketplace.atlassian.com/apps/1014681/zephyr-for-jira-testmanagement?hosting=cloud&tab=overview

Figura 7.4: Interface do Zephyr integrado a Jira.



Fonte: https://www.getzephyr.com/products/zephyr-for-jira

TestiTool[11]**:** É uma ferramenta desenvolvida usando PHP para gerenciamento de teste de software. Ele também possui código aberto, igual ao TestLink, e permite a criação de casos de testes e vinculá-los a um plano de teste desejado. Além disso, ele permite o gerenciamento dos requisitos funcionais e permite a vinculação desses requisitos com os casos de testes existentes na base de dados. A ferramenta também permite uma fácil exportação do plano de teste desejado para o formato .CSV.

O quadro comparativo na tabela 7.1 apresenta as vantagens do T-AADSP em relação às ferramentas apresentadas. Contudo ele não permite a divisão de controle de acesso das funcionalidades pelos diversos tipos de usuários.

[11] http://www.majordojo.com/testitool/

Tabela 7.1: Quadro comparativo entre as principais ferramentas de teste de *software* e a T-AADSP.

CARACTERÍSTICA	1	2	3	4	5
Permite relacionamento com casos de testes com um plano de teste	S	S	S	S	S
Permite selecionar planos de testes e executá-los dentro um ciclo de teste	S	S	S	S	S
Permite o cadastro de *bugs* encontrados durante o ciclo e a sua vinculação ao plano ou funcionalidade	S	N	S	N	N
Gera, de acordo com as informações do projeto e dos requisitos do sistema, uma priorização dos casos de testes mais urgentes de se executar	S	N	N	N	N
Permite geração de relatório/base de dados com todos os testes executados e *bugs* encontrados	S	S	S	S	S
Permite dividir o controle de acesso das funcionalidades pelos tipos de usuário	N	S	S	S	S

Fonte: Própria do autor.

Ferramentas – (1) T-AADSP-Test; (2) TestLink;(3) ALM; (4) Zephyr for Jira; (5) TestiTool.

Avaliação – 1. (S) Satisfaz o requisito em questão; 2. (N) Não atende o requisito; 3. (P) Atende Parcialmente o requisito; 4. (NA) O requisito em questão não foi avaliado; 5. (NE) A informação para avaliação do requisito Não foi Encontrada.

7.2 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.

7.2.1 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)*.

7.2.2 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 biblioteca que disponibiliza um conjunto de fontes e ícones 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.*

7.2.3 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 [12]. Um vídeo com o resumo das funcionalidades também pode ser encontra-lo no *YouTube*[13]. 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.

7.2. A FERRAMENTA T-AADSP *TEST*

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

[12] https://github.com/vitorssantana/T-AADSP-Test

[13] https://www.youtube.com/watch?v=3ETGsltNa1k

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 alterados, 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 já 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 cálculo 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 início, 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 possível 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 estiver sendo feita, o botão que permite a priorização fica bloqueado.

CAPÍTULO 05

CONCLUSÃO

8. CONCLUSÃO

Esse livro apresentou o gerenciamento de testes sob o ponto de vista do guia AADSP usando as Redes Bayesianas para priorização e predição. Para difusão do conhecimento nas indústrias de *software* foram discutidos os principais aspectos do teste de *software* e as principais atividades que o compõe sendo a principal o gerenciamento de priorização dos testes. Além disso, o gerenciamento de testes promove a identificação, controle de mudanças e importância dos mesmos como atividades centrais. Sendo a priorização a atividade mais relevante por conseguir definir uma ordem de realização das teses entre diversos requisitos e projetos.

Essa gerência difunde uma visão uniforme e linear a todos os interessados nos testes dos *softwares,* desde os gerentes de projetos, testadores aos clientes. A aplicação das Redes Bayesianas como sistema de representação do conhecimento para priorização de testes deve variar de empresa para empresa e de projeto para projeto de acordo com o contexto socioeconômico-técnico-cultural na qual será aplicada.

Para comprovar o atendimento das premissas estabelecidas nos projetos e na equipe, o guia AADSP baseia-se na entrega de artefatos como forma de validação e comprovação frente ao *software* que fora desenvolvido. Para a modelagem do conhecimento foram elencados diversos artefatos sendo que para cada um deles foi apresentado os campos (atributos) que o compõe bem como o grau de importância.

Além disso, para que os aderentes possam saber em qual nível o artefato encontra-se e como podem melhorar o mesmo, uma nota deve ser aplicada pelo auditor ao final do processo de auditoria. Além disso, o guia AADSP acredita que a geração de dados e *DataSets* com base nos requisitos, importância do projeto, nível dos profissionais da equipe desenvolvedora e dos testadores de uma organização pode auxiliar no processo de melhoria dos processos e contribuição para a área de *SearchBased Software Engeneering* com a aplicação de algoritmos genéticos tais como o *The Next Release Problem* para definir a priorização de teste dos requisitos mais adequados para o negócio do *software* e seus *stakeholders*.

O processo de auditoria dos artefatos tem como objetivo checar se a empresa cumpriu todos os critérios estabelecidos pelos artefatos da gerência. Para a avaliação de cada artefato previsto pelo guia AADSP uma meta foi criada com base no método GQM e um conjunto de questões avaliativas deve ser respondido baseado na Escala *Likert* com as notas de 1 a 5 para saber se o artefato atingiu ou não a meta e em quais questões especificas o artefato pode ser melhorado. Com todas as questões respondidas, o auditor pode inferir uma nota ao artefato baseado no somatório das questões avaliativas divido pela quantidade de questões.

Espera-se que com a entrega dos artefatos e o atendimento as questões avaliativas estabelecidas pelo mesmo as empresas aderentes ao guia AADSP consigam melhorar o seu processo de *software*, proporcionar uma melhor qualidade aos seus produtos e satisfação ao seu cliente final.

Para a priorização de testes, os artefatos foram criados visando fomentar uma base sólida que fornecesse apoio as demais gerências estabelecidas pelo guia AADSP. O livro AADSP Gerencia de Requisitos foi publicado anteriormente e outros posteriormente serão publicados para cada uma das outras gerências previstas pelo guia AADSP como: projetos, colaboradores, reuso e configuração e mudança.

BIBLIOGRAFIA

[1] “Conceitos de teste de software.” https://www. desenvolvedormatteus.com.br/conceitos-testes-software/. Accessed: 2018-08-01.

[2] 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.

[3] A. Onisko, M. J. Druzdzel, H. Wasyluk, *et al.*, “A bayesian network model for diagnosis of liver disorders,” in *Proceedings of the Eleventh Conference on Biocybernetics and Biomedical Engineering*, vol. 2, pp. 842–846, Citeseer, 1999.

[4] F. S. Costa *et al.*, “Aprendizagem estrutural de redes bayesianas pelo método de monte carlo e cadeias de markov,” 2013.

[5] A. I. F. Ferreira, G. Santos, R. Cerqueira, M. Montoni, A. Barreto, A. O. S. Barreto, and A. R. Rocha, “Applying iso 9001: 2000, mps. br and cmmi to achieve software process maturity: Bl informatica’s pathway,” in *29th International Conference on Software Engineering (ICSE’07)*, pp. 642–651, IEEE, 2007.

[6] A. I. F. Ferreira, R. Cerqueira, G. Santos, M. Montoni, A. Barreto, and A. R. Rocha, “Iso 9001: 2000, mps. br nível f e cmmi nível 3: Uma estratégia de melhoria de processos na bl informática,” *V SBQS*, pp. 375–382, 2006.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] L. R. P. Izabel, “Testes automatizados no processo de desenvolvimento de softwares,” tech. rep., Universidade Federal do Rio de Janeiro, 2014.

[15] International Software Testing Qualifications Board, *Certified Tester Foundation Level Syllabus*, 2011.

[16] P. C. Bernardo and F. Kon, “A importância dos testes automatizados controle ágil, rápido e confiável de qualidade,” *Engenharia de Software Magazine*, 2008.

[17] D. V. Pereira, “Estudo da ferramenta selenium ide para testes automatizados de aplicações web,” tech. rep., Universidade Estadual de Maringá, 2012.

[18] “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.

[19] 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 - N^{o}9*, december 2012.

[20] “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.

[21] A. Neto and D. Neto, “Introdução a teste de software,” 08 2018.

[22] “Tdd: fundamentos do desenvolvimento orientado a testes.” https://www.devmedia.com.br/ tdd-fundamentos-do-desenvolvimento-orientado-a-testes/ 28151. Accessed: 2019-07-02.

[23] M. Aniche, *Test-driven development: teste e design no mundo real*, vol. 1. Editora Casa do Código, 2014.

[24] “Desenvolvimento orientado por comportamento (bdd).” https://www.devmedia.com.br/ test-driven-development-tdd-simples-e-pratico/18533. Accessed: 2019-07-02.

[25] “Desenvolvimento orientado por comportamento (bdd).” https://www.devmedia.com.br/ desenvolvimento-orientado-por-comportamento-bdd/21127. Accessed: 2019-07-02.

[26] 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.

[27] 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.

[28] P. R. A. Firmino, “Redes bayesianas para a parametrização da confiabilidade em sistemas complexos,” 2004.

[29] A. Souza, “Redes bayesianas: Uma introdução aplicada a credit scoring,” 2010.

[30] S. Pena, “Thomas bayes, o ‘cara’!,” *Revista Ciência Hoje, vol. 38, pp. 22-29*, 2006.

[31] 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.

[32] “Teorema de bayes.” https://edisciplinas.usp.br/pluginfile.php/3110753/mod_resource/content/1/aula2.pdf. Accessed: 201907-02.

[33] 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.

[34] S. Sun, C. Zhang, and G. Yu, “A bayesian network approach to traffic flow forecasting,” *IEEE Transactions on intelligent transportation systems*, vol. 7, no. 1, pp. 124–132, 2006.

[35] 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.

[36] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper, “The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks,” in *AIME 89*, pp. 247–256, Springer, 1989.

[37] C. P. Team, “Cmmi for development, version 1.3,” *Software Engineering Institute, Carnegie Mellon University*, 2010.

[38] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI for development: guidelines for process integration and product improvement*. Pearson Education, 2011.

[39] N. Ehsan, A. Perwaiz, J. Arif, E. Mirza, and A. Ishaque, “Cmmi/spice based process improvement,” in *2010 IEEE International Conference on Management of Innovation & Technology*, pp. 859–862, IEEE, 2010.

[40] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI guidlines for process integration and product improvement*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[41] K. El Emam and H.-W. Jung, “An empirical evaluation of the iso/iec 15504 assessment model,” *Journal of Systems and Software*, vol. 59, no. 1, pp. 23–41, 2001.

[42] H.-W. Jung and R. Hunter, “The relationship between iso/iec 15504 process capability levels, iso 9001 certification and organization size: an empirical study,” *Journal of Systems and Software*, vol. 59, no. 1, pp. 43–55, 2001.

[43] M. C. Paulk, “Analyzing the conceptual relationship between iso/iec 15504 (software process assessment) and the capability maturity model for software,” in *1999 International Conference on Software Quality*, 1999.

[44] F. J. Pino, M. T. Baldassarre, M. Piattini, and G. Visaggio, “Harmonizing maturity levels from cmmi-dev and iso/iec 15504,” *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, no. 4, pp. 279–296, 2010.

[45] S. Peldzius and S. Ragaisis, “Comparison of maturity levels in cmmi-dev and iso/iec 15504,” *Applications of Mathematics and Computer Engineering*, pp. 117–122, 2011.

[46] S. M. Garcia, “Evolving improvement paradigms: capability maturity models and iso/iec 15504 (pdtr),” *Software Process: Improvement and Practice*, vol. 3, no. 1, pp. 47–58, 1997.

[47] M. BR, “Mps.br - melhoria de processo do software brasileiro - guia geral mps de software,” 2012.

[48] AADSP, “Adaptive approach for deployment of software process.” http://www.labrasoft.ifba.edu.br/wp-content/uploads/2016/ 10/Guia-AADSP.pdf, 2016 (acessado Dezembro 14, 2017).

[49] SOFTEX, “MPS.BR: melhoria de processo do software brasileiro,” SOFTEX, 2012.

[50] A. C. S. C. D. M. Macedo, *AADSP - Gerência de Requisitos*. Ed. Ixtlan, first ed., 2018. São Paulo/SP.

[51] T. C. Kinnear and J. R. Taylor, “Marketing research: an applied approach,” McGraw Hill, 1991.

ÍNDICE REMISSIVO

Agência Brasileira ISBN
ISBN: 978-65-86230-02-4