UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA A PRIORIZAÇÃO DE CASOS DE TESTE DE REGRESSÃO BASEADA EM RASTREABILIDADE

GABRIEL GIOANNINI MALIMPENSA

ORIENTADORA: PROFA. DRA. SANDRA CAMARGO PINTO FERRAZ FABBRI

São Carlos - SP
Fevereiro/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA A PRIORIZAÇÃO DE CASOS DE TESTE DE REGRESSÃO BASEADA EM RASTREABILIDADE

GABRIEL GIOANNINI MALIMPENSA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.
Orientadora: Profa. Dra. Sandra Camargo Pinto Ferraz Fabbri.

São Carlos - SP
Fevereiro/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Gabriel Gioannini Malimpensa, realizada em 05/03/2018:

Profa. Dra. Sandra Camargo Pinto Ferraz Fabbri
UFSCar

Prof. Dr. Auri Marcelo Rizzo Vincenzi
UFSCar

Prof. Dr. Katia Romero Felizardo Scannavino
UTFPR

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Katia Romero Felizardo Scannavino e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

Profa. Dra. Sandra Camargo Pinto Ferraz Fabbri

Aos meus pais e à minha noiva

AGRADECIMENTOS

Aos meus pais Carlos e Roseli, pelo apoio incondicional e confiança desde sempre.

À minha noiva Eloisa, pela paciência, compreensão e pelo incentivo nas horas mais difíceis.

À minha orientadora Profa. Dra. Sandra Fabbri, pela orientação e dedicação.

Ao meu amigo Prof. Dr. André Di Thommazo, pela ajuda e pelo companheirismo.

Aos colegas Rodrigo Ramos, Ana Paula Lima, Paulo Henrique Santana, Estevão Coelho, Fábio Fontana e Rafael Chuque, pela contribuição.

Agradeço a todos que fazem parte da minha vida, família, amigos, professores, colegas de trabalho, que de maneira direta ou indireta contribuíram para a conclusão deste trabalho.

Se não puder voar, corra. Se não puder correr, ande.

Se não puder andar, rasteje, mas continue em frente de qualquer jeito.

Martin Luther King

RESUMO

Contexto: Construir software com qualidade respeitando custos e prazos é um dos grandes desafios da área de desenvolvimento de sistemas. A rastreabilidade de requisitos e a atividade de teste, como testes de regressão, são alternativas para superar esses desafios. Porém, é sabido que a atividade de teste é bastante custosa e toma grande parte do tempo de todo o ciclo de desenvolvimento. Por isso, justifica-se a importância da priorização de casos de teste, para que os defeitos sejam revelados o quanto antes. **Objetivo**: O objetivo deste trabalho foi propor uma abordagem para a priorização de casos de teste no contexto de testes de regressão, baseada em rastreabilidade. **Metodologia**: Foi realizado um levantamento bibliográfico por meio de um mapeamento sistemático com o intuito de identificar as principais técnicas de priorização de casos de teste. Foi desenvolvido um *plug-in* de priorização de casos de teste, baseado na abordagem proposta, para uma ferramenta de ampla utilização na indústria (Jira). Também foram conduzidos estudos de caso com sistemas reais a fim de avaliar a abordagem proposta. **Resultados**: A abordagem elaborada neste trabalho apresentou boa eficácia nos estudos experimentais conduzidos. Em um dos estudos de caso, a priorização determinada pela abordagem apresentou melhor eficácia que a priorização feita manualmente por analistas de qualidade e também foi capaz de encontrar mais defeitos. **Conclusões**: Os resultados obtidos mostram que a abordagem elaborada pode ser uma alternativa para a aplicação da priorização de casos de teste na indústria.

Palavras-chave: priorização de casos de teste, testes de regressão, rastreabilidade de requisitos.

ABSTRACT

Context: Build software with quality while respecting cost and time is one of the great challenges of systems development. The requirements traceability and testing activity, such as regression tests, are alternatives to overcome these challenges. However, it is known that the testing activity is very costly and takes too much time for the entire development cycle. Therefore, it justifies the importance of prioritizing test cases, so that the defects are revealed as soon as possible. **Objective**: The objective of this work was to propose an approach for prioritizing test cases in the context of regression testing, based on traceability. **Methodology**: A literature review was conducted through a systematic mapping in order to identify the main techniques for prioritizing test cases. It was developed a plug-in for prioritization of test cases based on the proposed approach, for a widely used industry tool (Jira). It was also conducted case studies with real systems to evaluate this approach. **Results**: The approach elaborated during this work showed good efficacy in the conducted experimental studies. In one of the case studies, the prioritization determined by the approach presented better efficiency than the prioritization manually done by quality analysts and it also was able to find more defects. **Conclusions**: The obtained results show that the elaborated approach can be an alternative for the application of the prioritization of test cases in the industry.

Keywords: test case prioritization, regression tests, requirements traceability.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APFD – *Average Percentage of Faults Detected*

API – *Application Program Interface*

CASE – *Computer-Aided Software Engineering*

REST – *Representational State Transfer*

RTM – *Requirements Traceability Matrix*

RTS – *Regression Test Selection*

SOAP – *Simple Object Access Protocol*

SSL – *Secure Socket Layer*

TCP – *Test Case Prioritization*

TSM – *Test Suite Minimization*

UML – *Unified Modeling Language*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Este capítulo contextualiza o problema tratado nesta dissertação, apresenta a motivação e os objetivos que se desejam alcançar, a metodologia utilizada para o desenvolvimento do trabalho e a organização do texto.

1.1 Contexto

Utilizar e interagir com os mais variados tipos de software já faz parte do cotidiano de muitas pessoas, seja no trabalho ou em atividades comuns. Porém, desenvolver software é bastante complexo e há vários desafios a serem enfrentados pela indústria, como atender às mudanças nos requisitos, custos, prazos e garantir qualidade (PRESSMAN; MAXIM, 2015; SOMMERVILLE, 2015). Conseguir superar esses desafios e sanar as necessidades dos clientes é essencial para que uma empresa consiga se manter no dinâmico e competitivo mercado de software.

O Standish Group (2018) faz periodicamente, desde 1994, um relatório que apresenta a taxa de sucesso de projetos de software. O relatório, denominado *CHAOS Report*, do ano de 2015 avaliou 50.000 projetos em todo o mundo, e apenas 29% destes projetos foram concluídos com sucesso. Dos projetos que foram concluídos com sucesso, somente 8% eram considerados grandes ou muito grandes, enquanto 62% eram considerados pequenos. Esses dados sugerem que quanto maior e mais complexo um projeto, menor é a sua taxa de sucesso.

Por outro lado, existem atividades do processo de desenvolvimento de software como o planejamento das tarefas, o gerenciamento dos requisitos e a garantia da qualidade, que são muito importantes. A ausência ou insuficiência de

tais atividades pode contribuir ou ocasionar o fracasso de um projeto, principalmente os de grande porte (CASTOR; PINTO; SILVA; CASTRO, 2004; PRESSMAN; MAXIM, 2015; SOMMERVILLE, 2015).

Para tanto, o uso da rastreabilidade no gerenciamento dos requisitos e o processo de teste são alternativas para superar alguns desafios da área. Segundo Cleland-Huang, Gotel e Zisman (2012), a utilização da rastreabilidade de requisitos facilita a análise de impacto das mudanças e dá suporte aos testes de regressão. Da mesma forma, Pressman e Maxim (2015) indicam que as atividades de teste são essenciais para qualquer empresa que faz produtos para outras pessoas.

Mesmo tendo essa grande importância, o processo de teste pode ser muito custoso ou até inviável, dependendo do tamanho do conjunto de casos de teste existente (ABID; NADEEM, 2017). Por isso, grandes autores da área de teste de software como Rothermel et al. (1999, 2001) e Elbaum et al. (2000, 2001, 2002) destacam a importância da priorização de casos de teste. O intuito da priorização é encontrar o maior número de defeitos o mais rápido possível, ou seja, logo nos primeiros casos de teste. Assim, é possível manter a qualidade dentro de um prazo factível.

Levando em consideração o contexto apresentado anteriormente, o Laboratório de Pesquisa em Engenharia de Software (LaPES) propõe soluções para diminuir as dificuldades da área, como a técnica TUCCA (*Technique for Use Case Construction and Construction-Based Requirements Analysis*), que visa sistematizar a construção do Modelo de Casos de Uso e a inspeção do Documento de Requisitos (BELGAMO; FABBRI, 2005), e a ferramenta COCAR, que dá suporte a algumas atividades do processo de desenvolvimento de software (DI THOMMAZO, 2007).

A ferramenta COCAR foi elaborada por Di Thommazo (2007) e Martins (2007) em seus estudos para defesa de trabalho de mestrado. A primeira versão da ferramenta COCAR era responsável pelo gerenciamento de requisitos e pela estimativa do tamanho do software. A partir do ano de 2010 a ferramenta foi remodelada e novos estudos foram iniciados com foco em rastreabilidade de requisitos. O autor deste trabalho participou do desenvolvimento dessa nova versão em um projeto de iniciação científica durante todo o ano de 2011. A ferramenta sofreu evoluções e passou a ser capaz de gerar matrizes de rastreabilidade de requisitos por meio de diferentes abordagens, fazendo uso de técnicas de processamento de linguagem natural e inteligência computacional (DI THOMMAZO

et al., 2014b). O autor deste trabalho de mestrado ajudou na implementação de duas das quatro abordagens propostas (DI THOMMAZO et al., 2012).

Este trabalho é uma continuação dos estudos feitos por Di Thommazo (2007, 2012, 2013a, 2013b, 2014a, 2014b, 2015) e também envolve a rastreabilidade de requisitos.

A motivação e os objetivos são apresentados na próxima seção.

1.2 Motivação e Objetivos

Conforme mencionado anteriormente, a aplicação da estratégia de testes de regressão é muito importante para a garantia da qualidade, mas pode ser muito custosa dependendo do tamanho do conjunto de casos de teste de um projeto. Por isso, a priorização dos casos de teste é uma alternativa viável para que se possam aplicar os testes de regressão em um tempo factível (ELBAUM; MALISHEVSKY; ROTHERMEL, 2002).

Motivado por esse cenário, a solução proposta por este trabalho para a aplicação dos testes de regressão integra uma abordagem de priorização de casos de teste baseada em rastreabilidade e um recurso automatizado na forma de um *plug-in* para o Jira, com suporte à utilização da abordagem de priorização. O Jira é uma ferramenta robusta utilizada por mais de 11500 organizações em todo o mundo e entre outras funcionalidades, ela cobre todo o ciclo de desenvolvimento de software. Além disso, o Jira permite customizações e disponibiliza API para desenvolvimento.

Sendo assim, os principais objetivos deste trabalho são:

- Proposição de abordagem para priorização de casos de teste no contexto de testes de regressão, baseada em rastreabilidade de requisitos;
- Implementação de *plug-in* com suporte à abordagem proposta, para a ferramenta Jira;
- Avaliação da abordagem proposta por meio de estudos experimentais com sistemas reais;

- Medição da eficácia da abordagem proposta por meio de uma métrica bastante utilizada por trabalhos da área, denominada *Average Percentage of Faults Detected* (APFD);
- Comparação da eficácia da abordagem proposta com outras abordagens encontradas na literatura.

Na próxima seção é mostrada a metodologia de trabalho adotada para se alcançar os objetivos apresentados acima.

1.3 Metodologia de Desenvolvimento do Trabalho

Inicialmente foi feito o levantamento da bibliografia relacionada ao trabalho, que envolve a priorização de casos de teste. Para esse levantamento bibliográfico, foi elaborado um planejamento para a pesquisa utilizando o paradigma GQM (*Goal, Question, Metric)* (BASILI; CALDIEIRA; ROMBACH, 1994). Esse levantamento foi conduzido por meio de um mapeamento sistemático seguindo a proposta de Petersen et al. (2008). Os resultados desse estudo são descritos na Seção 3.2.3 desta dissertação. Após a condução desse mapeamento, também foi feito um estudo da abordagem de seleção de casos de teste proposta por Di Thommazo et al. (2015). Essa abordagem foi usada como base para o desenvolvimento da abordagem de priorização de casos de teste deste trabalho.

Após o levantamento bibliográfico e o estudo da abordagem de Di Thommazo (2015), foi feita a proposição da abordagem para a priorização de casos de teste, baseada em rastreabilidade.

Além da proposição da abordagem de priorização, também decidiu-se implementar uma ferramenta que permitisse a utilização da abordagem proposta. Isso ocorreu porque durante a condução do levantamento bibliográfico, foram encontradas poucas ferramentas que dão suporte à priorização de casos de teste. E ainda, grande parte das que foram encontradas na literatura são ferramentas com intuito acadêmico, e não são usadas na indústria. Por isso, primeiramente foi desenvolvido um módulo para a ferramenta COCAR, mas como dito anteriormente, pela dificuldade das empresas em adaptar seu processo de desenvolvimento a ela,

foi desenvolvido um *plug-in* para a ferramenta Jira, que é bastante utilizada por diversas empresas da área.

O *plug-in* desenvolvido para o Jira tem o formato de uma aplicação web e está hospedado em servidores virtuais na nuvem. Esses padrões e outros requisitos necessários como o protocolo de comunicação com o Jira tiveram que ser estudados diretamente em sua documentação oficial (ATLASSIAN, 2017).

Por fim, foram conduzidos dois estudos de caso com o intuito de avaliar a abordagem de priorização e também o *plug-in*. Esses estudos de caso foram feitos com sistemas reais. O primeiro com um sistema de compartilhamento de mídias digitais contando com 13 requisitos e pouco mais de 30 casos de teste. O segundo estudo de caso foi realizado em parceria com a empresa Monitora, de São Carlos. O sistema do estudo de caso em questão foi o Medical Box, um grande sistema da área médica, responsável pelo gerenciamento de clínicas, médicos, funcionários e pacientes. Também cuida do agendamento de consultas e atendimentos. Esse sistema já está em produção e conta com 73 requisitos e mais de 100 casos de teste.

Ao final dos estudos experimentais, foram feitas as medições de eficácia da abordagem de priorização por meio da métrica APFD proposta por Elbaum, Malishevsky e Rothermel (2000). Ela mede a taxa de detecção de defeitos de um conjunto priorizado de casos de teste. Isso significa que quanto maior o valor do APFD, maior a eficácia do conjunto para encontrar defeitos. Dessa forma, também foi possível comparar os resultados obtidos sobre a eficácia da abordagem proposta com outras abordagens de priorização encontradas na literatura.

A próxima seção apresenta a organização desta dissertação de mestrado.

1.4 Organização do Trabalho

Este trabalho está dividido em seis capítulos. Este capítulo introdutório apresentou o contexto, a motivação e os objetivos, assim como a metodologia de trabalho adotada.

No Capítulo 2 é apresentada a fundamentação teórica. Entre os assuntos abordados estão teste de software, testes de regressão, priorização de casos de teste e rastreabilidade de requisitos.

O Capítulo 3 mostra toda a revisão sistemática da literatura realizada durante a pesquisa, englobando a priorização de casos de teste, tema principal desta dissertação.

O Capítulo 4 apresenta a abordagem de priorização de casos de teste que foi proposta e desenvolvida para este trabalho, e também o *plug-in* desenvolvido para a ferramenta Jira.

No Capítulo 5 são mostrados os estudos experimentais que foram conduzidos para a avaliação da abordagem de priorização, assim como os resultados obtidos.

Por fim, o Capítulo 6 apresenta as conclusões desta dissertação.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica que fornece todo o embasamento para a realização deste trabalho.

2.1 Considerações Iniciais

Como apresentado no capítulo anterior, a indústria de software enfrenta grandes desafios para garantir custos, prazo e qualidade. Nesse cenário, o uso da rastreabilidade no gerenciamento dos requisitos e o processo de testes se mostram como boas alternativas para superar esses desafios (CLELAND-HUANG; GOTEL; ZISMAN, 2012; PRESSMAN; MAXIM, 2015). Dessa forma, neste capítulo de fundamentação teórica do trabalho são abordados conceitos e técnicas de teste de software e rastreabilidade de requisitos.

Testes de regressão e priorização de casos de teste são os conceitos de teste de software que têm maior ênfase. A rastreabilidade também é assunto importante, pois foi utilizada na proposição da abordagem de priorização de casos de teste, no contexto de testes de regressão.

Este capítulo possui seis seções. Na primeira seção foram feitas as considerações iniciais. A segunda seção trata dos conceitos gerais de teste de software, enquanto a terceira apresenta testes de regressão e a quarta foca em priorização de casos de teste. A quinta seção faz uma breve introdução sobre requisitos de software e rastreabilidade de requisitos. A sexta seção encerra com as considerações finais.

2.2 Teste de Software

Teste é uma atividade muito importante dentro do ciclo de vida de desenvolvimento de um software. Testar um software consiste em executá-lo usando dados artificiais. Dessa forma é possível analisar os resultados dos testes em busca de erros ou anomalias (SOMMERVILLE, 2015).

Segundo Sommerville (2015), o processo de teste visa:

- Garantir que o software foi desenvolvido de acordo com os requisitos definidos;
- Descobrir situações em que o comportamento do software não está dentro do esperado.

A atividade de teste deve ser dividida em quatro etapas, as quais devem estar presentes em todo o ciclo de desenvolvimento do software (PRESSMAN; MAXIM, 2015). São elas:

1. Planejamento;
2. Projeto de casos de teste;
3. Execução;
4. Avaliação de resultados.

O planejamento da atividade de teste é tão importante quanto o planejamento de qualquer atividade da fase de desenvolvimento, e o resultado desse planejamento é o plano de teste. No plano de teste são estimados os recursos necessários para as demais etapas e são definidos os critérios e as técnicas que serão aplicadas (HASS, 2014; MALDONADO; FABBRI, 2001).

O projeto de casos de teste é de fato a etapa de criação dos casos de teste. Para isso, é necessário levar em consideração todas as condições levantadas no plano de teste. Os casos de teste também devem seguir os critérios e técnicas adotadas na etapa de planejamento (HASS, 2014; MALDONADO; FABBRI, 2001).

Na etapa de execução, todos os casos de teste são executados de acordo com o planejado. Conforme os defeitos vão sendo encontrados, eles devem ser

registrados com detalhes para facilitar a futura correção (HASS, 2014; MALDONADO; FABBRI, 2001).

Por fim, na etapa de avaliação de resultados são coletados todos os dados obtidos na etapa de execução. Também é analisado o desempenho da equipe, as métricas definidas anteriormente no planejamento e até as estimativas de recursos. O ideal é que, a partir de todas essas informações, sejam feitos ajustes no processo para as próximas execuções (HASS, 2014; MALDONADO; FABBRI, 2001).

2.2.1 Fases de Teste

O processo de teste acompanha o software desde o seu início. Da mesma forma que o ciclo de vida de um software é dividido em fases, iniciando em um nível mais alto de abstração até a codificação em si, o processo de teste também é dividido em fases. Cada fase engloba as etapas mencionadas anteriormente. De maneira geral, são três fases: teste de unidade, teste de integração e teste de sistema (BLACK, 2009; MALDONADO; FABBRI, 2001; PRESSMAN; MAXIM, 2015).

A Figura 2.1 apresenta as principais fases de teste e as correspondentes fases do ciclo de vida do software.

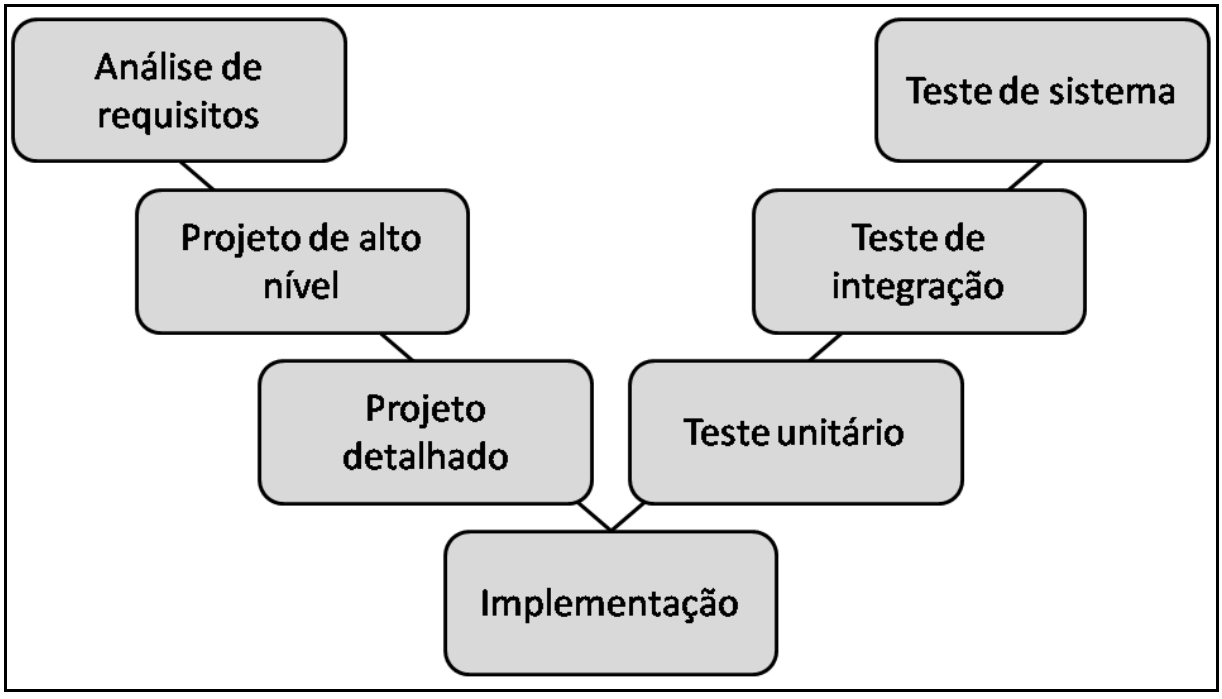


Figura 2.1 – Fases do processo de teste

O teste de unidade foca em descobrir erros de lógica e de implementação na menor unidade do projeto, podendo esta ser uma função, um método ou uma classe. Black (2009) diz que uma unidade é o menor trecho de código que faz sentido de maneira isolada. Geralmente os programadores testam o próprio código, mas pode ocorrer de um programador realizar os testes de unidade de outro programador. Também é conhecido que dois programadores podem trabalhar juntos tanto na codificação quanto na realização dos testes de unidade, técnica denominada programação em pares (BLACK, 2009).

O teste de integração tem por objetivo descobrir erros na integração das unidades, ou seja, nas interfaces entre elas (MALDONADO; FABBRI, 2001). Pressman e Maxim (2015) dizem que mesmo tendo testado todas as unidades do sistema separadamente e não ter encontrado defeitos, isso não quer dizer que não é necessário realizar os testes de integração. Unidades podem causar efeitos diferentes em outras unidades, que por sua vez podem apresentar um comportamento diferente do esperado. Segundo Black (2009), os fatores mais importantes para o sucesso da fase de testes de integração são o conhecimento e a habilidade do programador, ou seja, as atividades de teste de integração devem ficar com os programadores mais experientes.

Pressman e Maxim (2015) apresentam duas abordagens para realizar os testes de integração: *top-down* (de cima para baixo) e *bottom-up* (de baixo para cima). Na abordagem *top-down* os módulos do software vão sendo integrados de cima para baixo de acordo com a hierarquia de controle, iniciando pelo módulo principal. Há a necessidade da utilização de *stubs*, que são módulos falsos criados com o intuito de substituir os módulos ou unidades reais do sistema. Essa abordagem é mais indicada quando se deseja testar primeiramente os pontos de decisão e controle do software, que costumam estar nos níveis mais altos da hierarquia. Já a abordagem *bottom-up* inicia a integração com os módulos mais baixos da hierarquia, que são tidos como atômicos. Nesse caso não há a necessidade de utilizar *stubs*, pois os módulos subordinados já estarão criados (PRESSMAN; MAXIM, 2015).

O teste de sistema engloba o sistema como um todo, com todas as unidades e módulos integrados. Além de cobrir as funcionalidades, o teste de sistema também é responsável por exercitar outros fatores relativos ao sistema como segurança, desempenho e até mesmo implantação (PRESSMAN; MAXIM, 2015). O teste de

sistema tende a ser funcional ou comportamental porque, em geral, é executado por uma equipe de testadores e não de programadores. Dessa forma, a equipe não conhece a estrutura interna do sistema. Esse tipo de teste também é conhecido como teste caixa-preta (BLACK, 2009).

2.2.2 Técnicas e Critérios de Teste

Para garantir alta qualidade em um software é necessário que ele passe por um processo de teste. Porém, realizar o teste exaustivo, ou seja, testar todas as possibilidades de entradas é quase sempre impraticável. Então, para que o processo de teste seja viável e ainda garanta alta qualidade no software é preciso utilizar técnicas de teste. São elas: técnica estrutural, técnica funcional ou comportamental e técnica baseada em defeitos (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001; MYERS; SANDLER; BADGETT, 2011).

Segundo Myers, Sandler e Badgett (2011), para elaborar um rigoroso processo de teste é recomendado que se use uma combinação das técnicas, ou se possível todas elas, uma vez que cada uma tem suas vantagens e desvantagens.

A Figura 2.2 mostra uma representação da combinação das técnicas de teste.

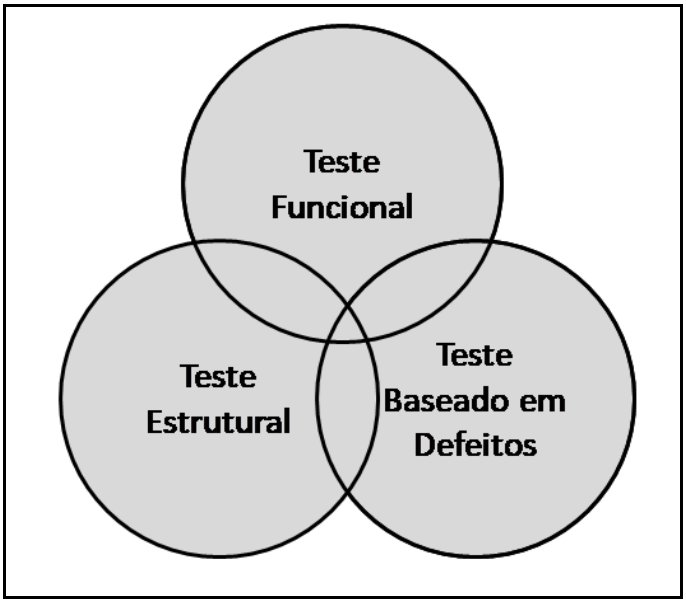


Figura 2.2 – Combinação das técnicas de teste

O teste estrutural, também conhecido como teste caixa-branca, é baseado no conhecimento da estrutura interna do programa que está sendo testado

(MALDONADO et al., 2004; MALDONADO; FABBRI, 2001). Seu objetivo é encontrar defeitos nos elementos estruturais de mais baixo nível, como o código fonte, por exemplo. Segundo Black (2009), é necessário um avançado conhecimento técnico do sistema, consequentemente, esses testes geralmente são planejados e executados pelos próprios programadores.

Os critérios do teste estrutural podem ser divididos em dois grupos principais: os baseados em fluxo de controle e os baseados em fluxo de dados (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001).

O teste estrutural baseado em fluxo de controle utiliza uma representação conhecida como grafo de fluxo de controle para auxiliar na elaboração dos casos de teste. Esse grafo é composto por nós que representam blocos de código e por arestas que representam os desvios entre os blocos. Dessa forma é possível usar um ou mais critérios para cobrir o código sob teste, uma vez que testar todos os caminhos pode se tornar inviável, ainda mais se este código tiver muitos laços de repetição (MYERS; SANDLER; BADGETT, 2011).

Os critérios mais utilizados para fluxo de controle são (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001):

- Todos-nós: esse critério exige que os testes exercitem ao menos uma vez todos os nós ou vértices do grafo, ou seja, todos os comandos;
- Todos-ramos: esse critério requer que cada aresta do grafo, que representa desvio de fluxo, seja executada ao menos uma vez;
- Todos-caminhos: esse critério, como mencionado anteriormente, exige que todos os caminhos possíveis do grafo sejam exercitados, mas isso pode não ser aplicável dependendo da complexidade do código.

Já o teste estrutural baseado em fluxo de dados segue o modelo de dados utilizado na implementação do programa. O principal critério para fluxo de dados é todos-usos, que consiste em elencar todas as ocorrências das variáveis do programa para gerar dados de teste. As ocorrências das variáveis dentro do programa podem ser classificadas de maneiras diferentes. As classificações possíveis são (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001):

- Definição: é quando ocorre uma atribuição de valor à variável;

- Uso computacional: é quando a variável é usada em uma expressão ou em um comando de saída;
- Uso predicativo: é quando a variável é usada em um predicado e acaba alterando o fluxo de controle.

O teste funcional ou comportamental, também conhecido como teste caixa-preta, é utilizado pelos testadores para encontrar defeitos em operações de alto-nível, como cenários e funcionalidades (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001). De maneira análoga ao teste caixa-branca, o teste caixa-preta requer grande conhecimento, porém do domínio da aplicação e das regras de negócio e não da parte técnica (BLACK, 2009). De acordo com Black (2009), bons testes funcionais são aqueles bem estruturados e que podem ser repetidos facilmente, assim como bons testes estruturais.

O teste funcional consiste em identificar as funcionalidades do software e criar casos de teste que sejam capazes de verificar se essas funcionalidades estão apresentando os resultados esperados (MYERS; SANDLER; BADGETT, 2011).

Dois critérios comumente usados do teste funcional são (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001):

- Particionamento de equivalência: esse critério divide o domínio de entradas do programa em entradas válidas e inválidas e a partir dessas classes seleciona os dados de entrada do caso de teste;
- Análise do valor limite: esse critério é um complemento ao critério anterior, focando nos limites das classes de equivalência.

Segundo Myers, Sandler e Badgett (2011), a grande vantagem do critério análise do valor limite em relação ao particionamento de equivalência é que o primeiro explora tanto os valores que estão dentro das classes de equivalência quanto os valores que estão em suas bordas.

O teste baseado em defeitos considera os erros mais comuns dos programadores para criar os casos de teste. O principal critério associado a essa técnica de teste é a análise de mutantes. O objetivo desse critério é avaliar o nível de qualidade dos casos de teste. Para se alcançar esse objetivo são realizadas alterações no programa, gerando assim os mutantes. Os casos de teste devem ser

capazes de identificar que o programa original e os programas mutantes apresentam comportamentos diferentes (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001).

Alguns exemplos comuns de alterações, que também são conhecidas como operadores de mutação:

- Retirada de comandos do programa;
- Troca de operadores relacionais;
- Troca de comandos, por exemplo, *while* por *do-while*;
- Interrupção proposital de laços de repetição;
- Troca de constantes.

A vantagem de usar o critério análise de mutantes está em provar a ausência de determinados defeitos, uma vez que ao eliminar o mutante prova-se que o programa original não possui aquele defeito. Por outro lado, a desvantagem é ser computacionalmente caro, ou seja, é necessário muito tempo e recursos para preparar e executar todos os mutantes (MALDONADO et al., 2004; MALDONADO; FABBRI, 2001).

2.3 Testes de Regressão

Teste de regressão é uma estratégia de teste de software que tem como objetivo garantir que alterações ou correções feitas em partes do software não alterem o funcionamento ou propaguem efeitos indesejados para outras partes (ABID; NADEEM, 2017; ANSARI et al., 2016; KAYES; ISLAM; CHAKARESKI, 2015).

Sempre que uma nova funcionalidade ou módulo é adicionado ao software, novos fluxos de dados são adicionados e novas lógicas de controle são estabelecidas, e isso pode causar problemas em funções ou módulos que anteriormente funcionavam de maneira correta (JEFFREY; GUPTA, 2008; PRESSMAN; MAXIM, 2015). Da mesma maneira, quando testes são aplicados com sucesso, defeitos são encontrados e devem ser corrigidos. Para fazer as correções também são necessárias alterações. É nesse contexto, com alguns cenários

diferentes que os testes de regressão estão inseridos (KLINDEE; PROMPOON, 2015; PRESSMAN; MAXIM, 2015; TYAGI; MALHOTRA, 2014).

Formalmente, testes de regressão podem ser definidos da seguinte maneira: dado um programa P_i e um conjunto de casos de teste T_i; quando P_i sofre alteração de versão para P_{i+1}, deve ser gerado um novo conjunto de casos de teste T_{i+1} que contem um subconjunto de T_i e novos casos de teste, capazes de testar as alterações em P_{i+1} (SRIKANTH; CASHMAN; COHEN, 2016).

Do (2016) mostra os avanços mais recentes em relação às técnicas de teste de regressão. Existem três técnicas que são comumente usadas para otimizar os testes de regressão: seleção de testes de regressão (RTS, do inglês *Regression Test Selection*), priorização de casos de teste (TCP, do inglês *Test Case Prioritization*) e minimização do conjunto de teste (TSM, do inglês *Test Suit Minimization*).

Técnicas de seleção de testes de regressão (RTS) selecionam um subconjunto de casos de teste que exercitam as partes que foram alteradas do software ou satisfaçam algum critério de seleção. Técnicas de priorização de casos de teste (TCP), que são alvo de estudo deste trabalho, reordenam os casos de teste para que eles sejam mais efetivos em encontrar defeitos. Técnicas de minimização do conjunto de teste (TSM) reduzem a quantidade de casos de teste do conjunto eliminando casos de teste redundantes, mas sempre com o intuito de manter a mesma capacidade de detecção de defeitos (DO, 2016; YOO; HARMAN, 2012).

Segundo Do (2016), a maioria das técnicas de RTS é baseada em código e usa informação de mudança e cobertura de código como critério de seleção dos casos de teste. Porém, outras fontes de informação podem ser usadas, como complexidade de código, dependência de código, histórico de defeitos.

As técnicas de TCP ajudam a revelar defeitos logo no início dos testes, e um grande diferencial é que todo o conjunto de casos de teste pode ser executado. Com isso, é possível evitar problemas relacionados à omissão de casos de teste, que é uma desvantagem das técnicas de RTS (DO, 2016). Técnicas de TCP são explicadas com detalhes na próxima seção.

As técnicas de TSM visam reduzir o tamanho do conjunto de casos de teste removendo os que estão obsoletos e redundantes. Isso ocorre porque o software evolui com o tempo, ou seja, novos componentes são adicionados e outros são alterados ou removidos, fazendo com que os casos de teste não sejam mais

adequados ou se tornem irrelevantes. A maioria das técnicas utiliza informação de cobertura de código para o processo de minimização (DO, 2016).

A Figura 2.3 mostra um esquema que exemplifica a aplicação das técnicas de seleção de testes de regressão, priorização de casos de teste e minimização do conjunto de teste. Do lado esquerdo está o programa P_i e o conjunto de casos de teste T_i. Do lado direito está o programa que sofreu alterações P_{i+1} e o conjunto de casos de teste de regressão T_{i+1} que deverá ser executado. Esse novo conjunto pode ser obtido por seleção, o qual seria equivalente ao conjunto T_i menos o conjunto de casos de teste que não exercitam as partes alteradas do código (T_i - $T_{\text{não exercita}}$); por priorização, o qual seria equivalente ao conjunto T_i reordenado por algum critério (T_i reordenado); ou por minimização, o qual seria equivalente ao conjunto T_i menos o conjunto de casos de teste redundantes (T_i - $T_{\text{redundante}}$).

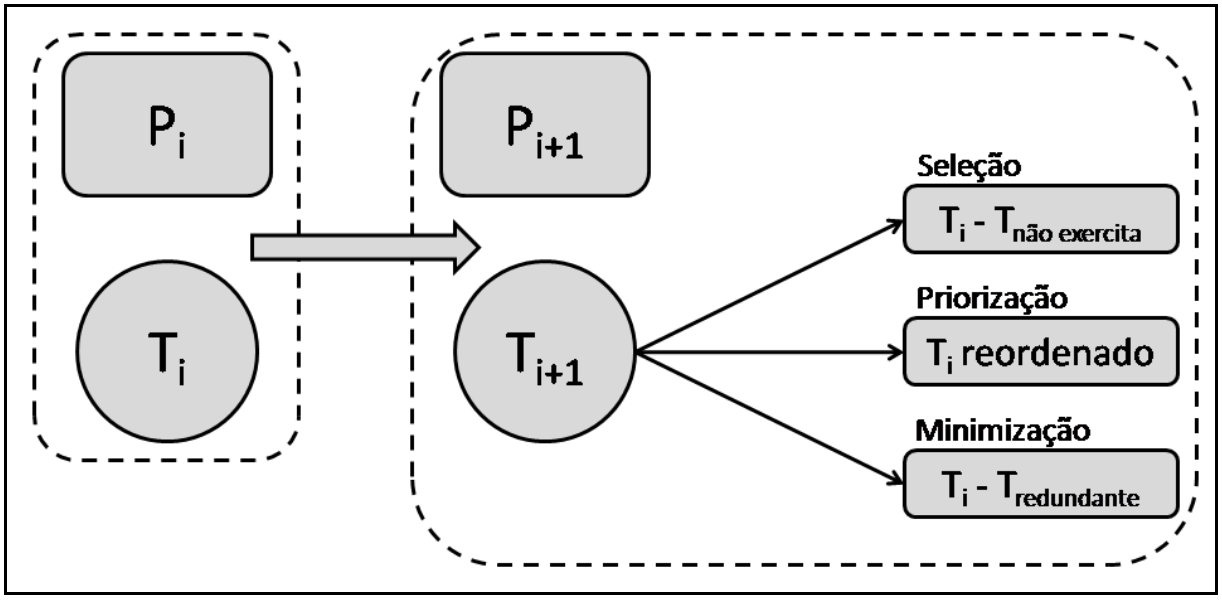


Figura 2.3 – Aplicação das técnicas RTS, TCP e TSM nos testes de regressão

Os testes de regressão podem ser executados manualmente ou por meio de uma ferramenta de apoio. Porém, executar todos os casos de teste pode demandar um esforço muito grande (KHIN; YOUNGSIK; JONG, 2008; ROONGRUANGSUWAN; DAENGDEJ, 2010; ROTHERMEL et al., 2001).

2.4 Priorização de Casos de Teste

Como mencionado na seção anterior, testes de regressão são aplicados para validar se alterações feitas no software não afetaram partes já testadas. Contudo, esse processo de teste é bastante custoso. Segundo Rothermel et al. (2001), um dos colaboradores de seus estudos reportou que para aplicar todos os casos de teste em um produto com aproximadamente 20.000 linhas de código, seria necessário um tempo de sete semanas. Portanto, engenheiros de testes costumam priorizar os testes de regressão, de forma que os testes mais importantes são executados primeiro (KRISHNAMOORTHI; SAHAAYA ARUL MARY, 2009; SIDDIK; SAKIB, 2014; VESCAN et al., 2017).

A priorização dos casos de teste é feita de acordo com algum critério. Por exemplo, um determinado critério pode definir uma ordem de casos de teste que seja eficaz para atingir um objetivo, mas que seja ineficaz para atingir outro objetivo (LIN et al., 2013; ROTHERMEL et al., 1999).

De acordo com Rothermel et al. (2001), alguns dos objetivos que os profissionais da área de testes buscam com a priorização dos casos de teste são:

- Aumentar a taxa de detecção de defeitos do conjunto de testes, revelando defeitos logo no início da execução dos testes;
- Aumentar a cobertura de código do sistema que está sendo testado, o mais rápido possível;
- Aumentar a taxa de detecção de defeitos de alto risco;
- Aumentar a probabilidade de revelar defeitos relacionados a mudanças no código.

Para alcançar tais objetivos existem diversas técnicas de priorização de casos de teste, que podem ser divididas em grandes grupos (CATAL; MISHRA, 2013; YOO; HARMAN, 2012):

- Técnicas baseadas em cobertura: prioriza os casos de teste de acordo com a cobertura de código (testes estruturais); casos de teste com maior cobertura podem aumentar a detecção de defeitos;

- Técnicas baseadas em distribuição: prioriza os casos de teste de acordo como seus perfis; casos de teste com o mesmo perfil são agrupados para evitar redundância;
- Técnicas baseadas em fatores humanos: prioriza os casos de teste de acordo com fatores que os testadores julgam mais importantes;
- Técnicas baseadas em probabilidade: prioriza os casos de teste de acordo com teorias probabilísticas; uma técnica bastante conhecida é a baseada em rede Bayesiana;
- Técnicas baseadas em histórico: prioriza os casos de teste de acordo com informações de histórico de execução dos casos de teste e mudanças no código;
- Técnicas baseadas nos requisitos: prioriza os casos de teste de acordo com informações extraídas dos requisitos como a importância do requisito para o cliente, a volatilidade do requisito, a complexidade de implementação do requisito;
- Técnicas baseadas em modelo: prioriza os casos de teste de acordo com informações extraídas de modelos UML (diagramas de sequência ou de atividades);
- Técnicas baseadas no custo dos casos de teste: prioriza os casos de teste de acordo com os custos definidos para cada caso de teste;
- Outras técnicas: outras técnicas conhecidas, mas que não entram em nenhum dos casos anteriores como priorização usando algoritmos genéticos, priorização dos casos de teste das partes de relevância, entre outros.

Todas essas técnicas foram levantadas e estudadas no mapeamento sistemático conduzido neste trabalho. O resultado desse estudo é apresentado na Seção 3.2.3.

Apesar dos vários tipos de técnicas de priorização e da grande quantidade de variações que os autores apresentam dentro de cada tipo, grande parte das técnicas utilizam a mesma métrica para avaliar os resultados e a sua eficácia. Essa métrica, chamada de *Average Percentage of Faults Detected* (APFD), foi proposta por Elbaum, Malishevsky e Rothermel (2000). Seus valores vão de 0 a 100 e quanto

maior o APFD, melhor a taxa de detecção de defeitos do conjunto de casos de teste. Consequentemente, mais rápido os defeitos são revelados por aquele conjunto. A fórmula para calcular o APFD é a seguinte:

$$APFD = 1 - \frac{(TF1 + TF2 + ... + TFn)}{nm} + \frac{1}{2n}$$

Na equação da métrica APFD, *n* representa o número de casos de teste do conjunto; *m* representa o número de defeitos revelados pelo conjunto; e *TFi* é o primeiro caso de teste da sequência que revelou o defeito *i*.

2.5 Requisitos de Software e Rastreabilidade de Requisitos

Um requisito de software pode ser: (1) uma função que o software deve realizar; (2) uma característica ou restrição que o software deve possuir; (3) uma condição que o software deve atender (GERACI et al., 1991; SOMMERVILLE, 2015). De forma geral, um requisito sempre visa atender uma necessidade do cliente (ABBOTT, 1986).

Segundo Sommerville (2015), os requisitos são classificados em:

- Requisitos funcionais: estão relacionados com a funcionalidade do sistema, ou seja, definem o que o sistema deve e/ou não deve fazer;
- Requisitos não funcionais: não estão relacionados com funcionalidade; em geral definem características ou condições que o sistema deve atender.

O levantamento e a análise de requisitos é a primeira etapa do ciclo de vida de um software. Porém, ao longo de todo o processo, os requisitos podem mudar (SOMMERVILLE, 2015). Existem vários motivos pelos quais os requisitos mudam, no entanto, conseguir se adaptar a mudanças é essencial, principalmente no mercado de software que é muito dinâmico.

Dessa forma, gerenciar os requisitos de um software para que se consiga maior agilidade de adaptação e de resposta às mudanças é fundamental. Um fator determinante para se alcançar essa agilidade é a rastreabilidade dos requisitos (CASTOR et al., 2004; GUO et al., 2009).

A rastreabilidade de requisitos é definida pelo acompanhamento e controle do ciclo de vida de um requisito (GUO et al., 2009). Esse controle deve estar presente desde o levantamento dos requisitos, fase de projeto, implementação e testes. A rastreabilidade permite identificar os relacionamentos existentes entre os próprios requisitos e dos requisitos com outros artefatos. Consequentemente, ela aumenta a qualidade do processo de software, pois fornece uma base para a evolução das mudanças nos requisitos (GUO et al., 2009).

Cleland-Huang, Gotel e Zisman (2012) mencionam que as principais vantagens na utilização de rastreabilidade de requisitos são: o acompanhamento da evolução do projeto, a análise de impacto das mudanças, a identificação da possibilidade de reuso, a diminuição de retrabalho e o suporte aos testes de regressão. Segundo Castor et al. (2004), a rastreabilidade é um fator muito importante no gerenciamento e na qualidade de projetos de software, e a sua falta pode causar atrasos e aumentar os custos em eventuais alterações ou correções no software.

Di Thommazo (2014a), em seus estudos sobre rastreabilidade de requisitos, propôs diferentes abordagens para gerar a Matriz de Rastreabilidade de Requisitos. Ela é uma matriz identidade que contém os valores de dependência entre cada requisito de um projeto. Uma dessas abordagens, chamada de RTM-E, utiliza os dados de entrada dos requisitos, que são os dados que eles manipulam, para calcular o valor de dependência entre eles.

Para calcular a dependência entre cada requisito, Di Thommazo (2014a) faz uso do Índice de Jaccard, que mede o nível de similaridade de dois conjuntos. Esse índice é obtido pela fração entre a intersecção dos dados dos conjuntos sobre a união dos dados dos conjuntos. Aplicando o índice ao cálculo de dependência entre os requisitos tem-se a fração entre a intersecção dos dados de entrada de dois requisitos pela união dos dados de entrada dos dois requisitos. O nível de dependência é expresso em porcentagem.

A Figura 2.4 apresenta um exemplo de dependência entre dois requisitos: o Requisito 1 (R1) possui os dados de entrada A, B e C, e o Requisito 2 (R2) possui os

dados de entrada C e D. Calculando a fração entre a intersecção dos dados de entrada de R1 e R2 (C) e a união dos dados de entrada de R1 e R2 (A, B, C, D), tem-se 25% de dependência.

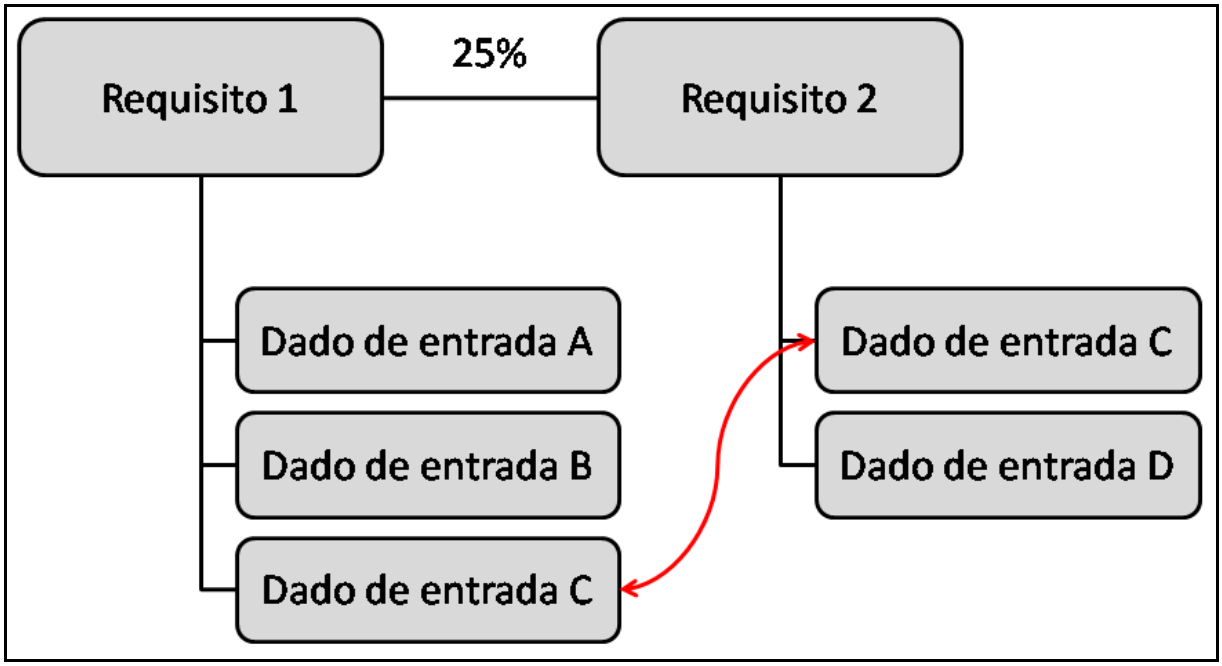


Figura 2.4 – Exemplo de dependência entre requisitos

A partir do valor obtido, Di Thommazo (2014a) classifica a dependência entre os requisitos em forte, fraca e inexistente.

2.6 Considerações Finais

Este capítulo apresentou a fundamentação teórica deste trabalho. Foram abordados os conceitos gerais de teste de software: os objetivos, as etapas da atividade de teste, assim como as fases de teste durante o ciclo de vida de um software e as principais técnicas com seus respectivos critérios. Também foi apresentado o conceito de testes de regressão e sua utilidade, e os conceitos de seleção de casos de teste, priorização de casos de teste e minimização do conjunto de casos de teste. Por fim, foram apresentados os conceitos de requisitos de software e a rastreabilidade entre eles.

Ainda relacionada à rastreabilidade de requisitos, foi apresentada a abordagem proposta por Di Thommazo (2014a) para calcular a dependência entre requisitos e consequentemente gerar a Matriz de Rastreabilidade de Requisitos baseada em dados de entrada (RTM-E). É importante frisar que, por ser uma continuação dos estudos de Di Thommazo (2007, 2012, 2013a, 2013b, 2014a, 2014b, 2015), este trabalho utiliza esse mesmo cálculo de dependência entre requisitos, e os valores obtidos são utilizados na priorização dos casos de teste.

No próximo capítulo é apresentada a revisão bibliográfica realizada para dar suporte a este trabalho. Será retomado o assunto de priorização de casos de teste, que está diretamente relacionado a este trabalho, e que será explorado com mais detalhes.

Capítulo 3

REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta a revisão bibliográfica realizada para esta dissertação de mestrado. Ela aborda o tema que envolve a proposta deste trabalho: priorização de casos de teste.

3.1 Considerações Iniciais

Como apresentado no Capítulo 1, este trabalho de mestrado propõe uma abordagem de priorização de casos de teste. O intuito de priorizar os casos de teste para aplicar a estratégia de testes de regressão é que os defeitos sejam revelados o quanto antes, logo nos primeiros casos de teste.

Por isso, para dar embasamento ao trabalho foi feito um mapeamento sistemático sobre priorização de casos de teste, a fim de investigar na literatura quais são as técnicas de priorização existentes, como é feita a validação dos estudos e quais são as ferramentas de apoio.

Este capítulo tem quatro seções. Na primeira seção foram feitas as considerações iniciais. A segunda seção apresenta o mapeamento sistemático conduzido sobre priorização de casos de teste. A terceira seção traz uma explanação sobre as duas ferramentas relacionadas a este trabalho: COCAR e Jira, e a quarta e última seção encerra com as considerações finais.

3.2 Mapeamento Sistemático sobre Priorização de Casos de Teste

Esta seção está dividida em três subseções. A primeira apresenta o planejamento realizado para a condução do mapeamento sistemático; a segunda apresenta a execução do mapeamento sistemático; a terceira e última apresenta uma discussão sobre os resultados obtidos.

3.2.1 Planejamento

Conforme mencionado anteriormente, o planejamento desta pesquisa foi feito de acordo com o paradigma GQM (BASILI; CALDIERA; ROMBACH, 1994). A revisão bibliográfica foi realizada por meio de um mapeamento sistemático, seguindo Petersen et al. (2008). Cada questão do paradigma GQM resulta em uma questão de pesquisa de um mapeamento sistemático, e cada métrica resulta em um atributo do formulário de extração de dados. A ferramenta StArt (FABBRI et al., 2012; HERNANDES et al., 2010; ZAMBONI et al., 2010), que também foi idealizada e desenvolvida no Laboratório de Pesquisa em Engenharia de Software (LaPES), foi utilizada como suporte computacional. A Figura 3.1 representa o uso do GQM no planejamento desta pesquisa e como ele foi utilizado para a condução do mapeamento sistemático.

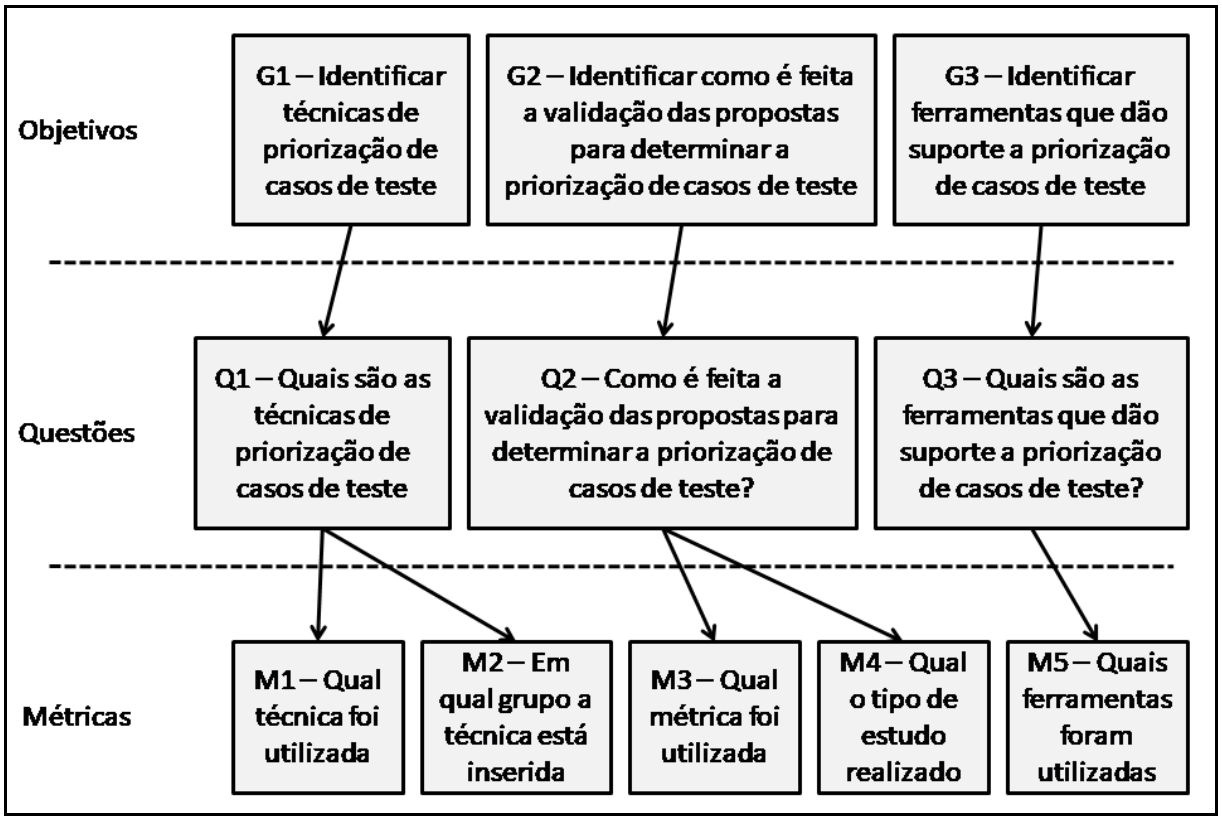


Figura 3.1 – Utilização de GQM no planejamento da pesquisa

Inicialmente esperava-se construir uma *string* de busca para cada questão planejada com o GQM, porém, durante esse processo percebeu-se que seria mais eficiente construir uma *string* genérica sobre priorização de casos de teste, pois dessa forma a busca retornaria resultados mais amplos que possivelmente responderiam às três questões planejadas.

3.2.2 Execução

A busca pelos estudos foi feita nas principais bases bibliográficas (ACM, IEEE, Scopus e Web of Science) através de uma *string* composta por palavras-chave. A *string* utilizada foi a seguinte: *("priorização" OR "prioritization") AND ("caso de teste" OR "casos de teste" OR "test case" OR "test cases") AND ("software")*. Destaca-se que essa busca foi feita em agosto de 2016 e atualizada em janeiro de 2018.

Inicialmente foram encontrados 348 estudos relacionados à priorização de casos de teste. Após a realização de uma seleção inicial, por título e resumo, foram

selecionados 259 estudos. Essa primeira seleção foi baseada em alguns critérios. O estudo que atendeu algum dos seguintes critérios foi selecionado:

- Apresenta uma técnica de priorização de casos de teste;
- Faz uma comparação entre técnicas de priorização de casos de teste;
- Apresenta uma ferramenta que apóia a priorização de casos de teste;
- Faz uma comparação entre ferramentas que apóiam a priorização de casos de teste.

Então, os estudos mais relevantes foram lidos e selecionados, para que algumas informações importantes para a pesquisa pudessem ser extraídas. Ao final foram aceitos 171 estudos, e os dados extraídos de cada um deles foram:

- A métrica utilizada;
- Se utiliza alguma ferramenta computacional;
- Qual o tipo de estudo apresentado;
- A técnica utilizada;
- Qual o grupo que a técnica se encaixa.

A partir da leitura dos estudos e da extração dos dados foi possível obter alguns resultados e chegar a algumas conclusões, que serão mostrados a seguir.

3.2.3 Discussão dos Resultados

Para comparar a eficácia das técnicas de priorização, a maioria dos estudos utiliza a métrica *Average Percentage of Faults Detected* (APFD), apresentada na Seção 2.4. Essa métrica, elaborada por Elbaum, Malishevsky e Rothermel (2000), é muito conhecida na área de priorização de casos de teste, sendo que estudos bastante significativos (INDUMATHI; SELVAMANI, 2015; KAYES; ISLAM; CHAKARESKI, 2015; KIM; BAIK, 2010; MEI et al., 2012; XIAO et al., 2017) sempre comparam a eficácia de suas técnicas com a de outras existentes na literatura utilizando APFD. Ainda com relação às métricas, um ano após a sua publicação, os mesmos autores publicaram um trabalho no qual ajustam a métrica para considerar

de forma ponderada o custo de cada um dos casos de testes. Esse estudo (ELBAUM; MALISHEVSKY; ROTHERMEL, 2001) foi base para as abordagens que consideram o tempo e custo da abordagem e serão detalhadas na sequência como abordagens *cost-aware.*

Outro fator importante levantado por este mapeamento sistemático é a utilização de ferramentas computacionais para dar apoio ao processo de priorização de casos de teste. Foi verificado que apenas 29% dos estudos apresentaram ou utilizaram uma ferramenta em conjunto com a abordagem ou técnica de priorização. Esse valor é baixo, se for considerado que para uma técnica ser de fato utilizada na indústria, muito provavelmente será por meio de uma ferramenta computacional. Muitos dos trabalhos apresentados utilizaram MATLAB, EXCEL, scripts ou protótipos para avaliar as abordagens propostas. Dentre as ferramentas utilizadas, a que obteve maior ocorrência foi a *Aristotle Program Analysis System*, com 12 ocorrências. Ela foi desenvolvida pelo grupo de Rothermel et al. e evoluiu nos demais trabalhos desse grupo de pesquisa. Caso semelhante ocorre com a segunda ferramenta mais referenciada, a ferramenta *Sofya*. Ela também está nos diversos estudos de Do et al. (2008) com 6 ocorrências.

Também foi notado que muitos estudos comparam os resultados obtidos de suas técnicas apenas com a técnica de priorização randômica. Alguns estudos (MOHAPATRA; PRASAD, 2014; PRAKASH; RANGASWAMY, 2012; THOMAS et al., 2014) indicam que essa comparação pode ser muito fraca para determinar se a técnica proposta tem um bom nível de eficácia. Sendo assim, esses trabalhos comparam suas respectivas técnicas de priorização com outras técnicas encontradas na literatura.

Com relação ao tipo de estudo utilizado para avaliar a abordagem proposta, somente 9,4% dos estudos foram feitos com estudo de caso em sistemas reais. A maioria dos estudos (79,5%) se tratava de estudo de caso em sistemas não reais, estudo de viabilidade ou ainda exemplo de uso das abordagens. Os demais trabalhos (11,1%) não apresentavam novas abordagens, pois comparavam abordagens ou apresentavam novas ideias ou caminhos para desenvolvimento futuro.

Foram identificadas diversas técnicas de priorização de casos de teste. Algumas são recorrentes em vários estudos e possuem propostas de evoluções ao longo do tempo como, por exemplo, as técnicas baseadas em cobertura e em

requisitos. Por outro lado, diversas técnicas encontradas na literatura são propostas isoladas e com propósito acadêmico.

Yoo e Harman (2012), que apresentam um *survey* sobre priorização de casos de teste de regressão, classificam as técnicas de priorização em grandes grupos, como mostrado na Seção 2.4. Segundo eles, a maioria das técnicas encontradas na literatura é baseada em cobertura. No estudo realizado, esse dado se confirma, uma vez que 26,3% das técnicas encontradas são baseadas exclusivamente em cobertura. A Figura 3.2 mostra um gráfico com a relação do número de estudos encontrados em relação aos grupos de técnicas de Yoo e Harman (2012).

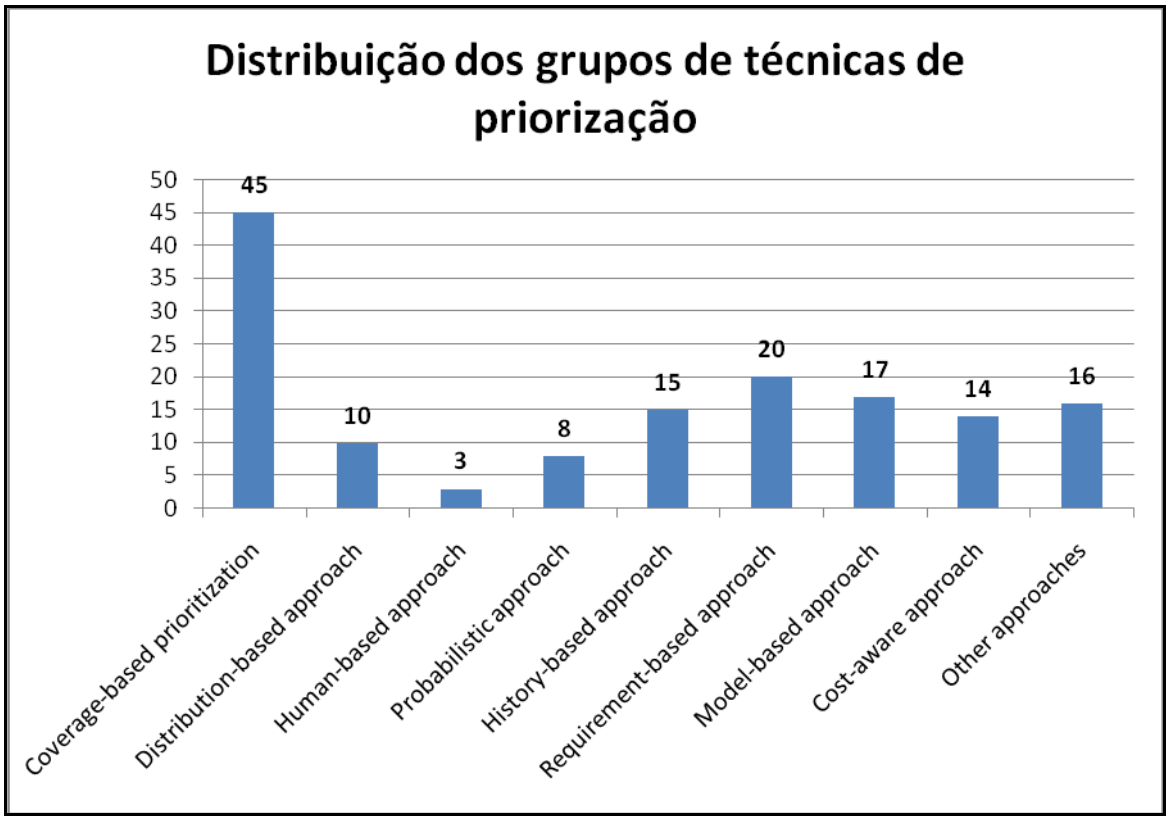


Figura 3.2 – Distribuição dos grupos de técnicas de priorização

Na sequência é apresentada uma descrição sobre as principais técnicas e os trabalhos mais relevantes de cada uma delas.

3.2.3.1 Coverage-based prioritization

A priorização baseada em cobertura é a mais utilizada e foi uma das primeiras a ser proposta na literatura. Os primeiros estudos a respeito dessa abordagem são

de Rothermel et al. (1999) e servem de base até para os estudos mais recentes. O princípio dessa abordagem é maximizar a cobertura do código fonte o quanto antes, com os casos de teste, para aumentar as chances de detectar defeitos.

Rothermel et al. (1999) apresentam algumas variações em sua abordagem. Essas variações estão relacionadas com a forma que é feita a cobertura do código fonte. São elas: *branch-total*, *branch-additional*, *statement-total*, *statement-additional*.

A técnica *branch-total* prioriza os casos de teste de acordo com número total de estruturas de decisão ou ramos encontrados no código, ou seja, os casos de teste que cobrem mais ramos serão executados primeiro. De maneira análoga, a técnica *statement-total* prioriza os casos de teste de acordo com o número total de instruções que eles cobrem.

Já as técnicas *branch-additional* e *statement-additional* são um pouco mais elaboradas, porque uma vez que um caso de teste que cobre determinados ramos (*branch-additional*) ou instruções (*statement-additional*) tenha sido executado é mais interessante que o próximo caso de teste a ser executado cubra outros ramos ou instruções que ainda não tenham sido exercitadas. Esse processo se repete até que todos os ramos ou instruções tenham sido exercitados por pelo menos um caso de teste.

Apesar das técnicas de cobertura adicional, como são chamadas, serem mais eficazes para maximizar a cobertura do código o mais cedo possível, elas são mais caras computacionalmente, porque precisam recalcular a informação de cobertura de cada caso de teste não priorizado, a cada nova execução.

Elbaum, Malishevsky e Rothermel (2000) estendem o trabalho anterior propondo técnicas que levam em consideração as funções do programa: *function-total* e *function-additional*. Essas técnicas são análogas às anteriores, porém calculam a quantidade total de funções e a quantidade de funções ainda não exercitadas, respectivamente.

Estudos mais recentes como o de Kwon et al. (2014), apresentam técnicas mais complexas, mas que de alguma forma, tiveram como base os estudos abordados anteriormente.

Kwon et al. (2014) descrevem uma técnica de priorização baseada em cobertura mas com conceitos de recuperação de informação. As informações são obtidas a partir do próprio código que está sendo testando, com o intuito de melhorar a priorização. São elas: *Term Frequency* (TF) e *Inverted Document Frequency* (IDF).

Basicamente, a primeira representa quantas vezes um termo aparece em um documento e a segunda representa o inverso da informação *Document Frequency* (DF), que representa o número de documentos de uma coleção que contém determinado termo. Resumidamente, se um termo tem um alto IDF, quer dizer que esse termo não aparece muito nos outros documentos e que ele é relativamente único no documento em questão.

Com essas duas informações (TF e IDF), a abordagem propõe o cálculo de um índice de similaridade que será usado para priorizar os casos de teste. A priorização considera que os elementos menos exercitados têm mais chance de apresentar defeitos, consequentemente, os casos de teste que cobrem mais elementos deste tipo são executados primeiro. De acordo com os resultados do estudo, essa técnica é mais eficaz que as técnicas comuns de cobertura, tendo um ganho no APFD de 4,7%.

Vedpal, Chauhan e Kumar (2015) apresentam uma técnica de priorização baseada na hierarquia de software orientado a objetos. Essa técnica parte da premissa que o conceito de herança facilita o reuso, porém pode propagar erros para as classes derivadas. Dessa forma, a técnica faz a priorização em dois níveis: o primeiro prioriza as classes de forma que as que tenham mais dependentes sejam testadas primeiro e o segundo, que prioriza os casos de teste para cada classe de acordo com a abordagem de cobertura. Como resultado, este estudo apresentou APFD melhor que a priorização randômica.

3.2.3.2 Distribution-based approach

Esse tipo de abordagem de priorização leva em consideração os perfis dos casos de teste. A partir de alguma métrica, os casos de teste podem ser agrupados de acordo com suas similaridades (YOO; HARMAN, 2012). Agrupar os casos de teste pode revelar informações importantes, como por exemplo:

- Casos de teste de um mesmo grupo podem ser redundantes;
- Grupos isolados podem conter casos de teste que induzem a comportamentos incomuns, que consequentemente têm mais chances de gerar defeitos.

Um dos primeiros estudos envolvendo a priorização baseada em distribuição foi o de Leon e Podgurski (2003), que elege um caso de teste por grupo e então os prioriza de acordo com a ordem de criação dos grupos.

Outro trabalho bastante interessante que utiliza essa abordagem é o de Carlson, Do e Denton (2011). Eles combinam o agrupamento dos casos de teste com outras abordagens. O agrupamento dos casos de teste é feito por similaridade e depois a priorização de cada grupo pode ser feita por quatro abordagens diferentes: por cobertura, por complexidade de código, pela informação do histórico de defeitos ou por uma combinação das duas últimas.

3.2.3.3 Human-based approach

Essa abordagem leva em consideração informações providas por pessoas da área de testes que estão envolvidas com o projeto (YOO; HARMAN, 2012). Malz e Gohner (2011) apresentam uma técnica bastante complexa que usa agentes inteligentes de software e lógica *fuzzy* para priorizar os casos de teste. Uma das fontes de dados para os agentes é o conhecimento da própria equipe de testes, como por exemplo, a complexidade de cada módulo do software.

3.2.3.4 Probabilistic approach

Nessa abordagem são aplicadas diferentes teorias probabilísticas para priorizar os casos de teste (YOO; HARMAN, 2012). Mirarab e Tahvildari (2007, 2008) apresentam uma abordagem baseada em Redes Bayesianas. A rede é construída a partir dos seguintes dados extraídos: mudanças nos elementos do programa, propensão de falhas nos elementos do programa e probabilidade de cada caso de teste revelar defeitos.

3.2.3.5 History-based approach

Essa abordagem utiliza informações do histórico de execução dos casos de teste, do histórico de defeitos ou do histórico dos próprios artefatos de software. Um dos estudos mais significativos desse tipo de abordagem é o de Kim e Porter (2002). Eles apresentam uma técnica que utiliza informações dos ciclos de teste anteriores para melhorar a priorização dos próximos ciclos. Para cada ciclo de teste, dois passos são realizados: (1) é dado um peso para cada função de acordo com a

frequência que ela é coberta por um caso de teste, de forma que as funções que são exercitadas com menos frequência ganham um peso maior; (2) definir o valor do histórico do teste (H_{tc}), que é o conjunto de execuções prévias do caso de teste *tc*. Dessa forma, os casos de teste que tiveram o maior peso ao longo das execuções (menor frequência), terão maior prioridade para o próximo ciclo de execução.

Os estudos de Fazlalizadeh et al. (2009) e Khalilian, Azgomi e Fazlalizadeh (2012) propõem melhorias na técnica de Kim e Porter. Segundo eles, um fator muito importante para a priorização é a capacidade de detecção de defeito de um caso de teste ao longo de sucessivas execuções. Por exemplo, se um caso de teste encontra 9 defeitos em 18 ciclos de execução, ele deve ter uma prioridade maior que um caso de teste que encontra 3 defeitos em 20 ciclos de execução. Ainda segundo eles, o número de ciclos em que um caso de teste é executado e o número de ciclos que aquele caso de teste revelou defeitos não devem ser tratados separadamente. Outro fator muito importante é o período em que um caso de teste não é executado. Eles defendem a ideia de que é preciso garantir que após alguns ciclos de execução, todos os casos de teste devem ser executados.

3.2.3.6 Requirement-based approach

Essa abordagem leva em consideração informação dos próprios requisitos do sistema para realizar a priorização dos casos de teste. Srikanth, Williams e Osborne (2005) propuseram essa abordagem. Eles também desenvolveram a ferramenta PORT (*Prioritization Of Requirements for Testing*) para dar suporte à abordagem proposta, a qual analisa quatro fatores associados aos requisitos: prioridade dos requisitos atribuída pelo cliente, complexidade de implementação na visão do desenvolvedor, volatilidade do requisito e propensão à falha dos requisitos. Com esses valores definidos é possível calcular o fator de priorização do caso de teste. Srikanth et al. (2012, 2014, 2016) ainda contribuiu com outros estudos relativos à abordagem baseada em requisitos e também propôs algumas melhorias em sua técnica. Outro trabalho bastante significativo é o de Krishnamoorthi e Mary (2009), que apresenta novos fatores para priorização: mudanças nos requisitos, impacto das falhas, integridade e rastreabilidade. Nesse trabalho a rastreabilidade considerada é a existente entre os requisitos e os casos de testes. Para a proposta de priorização dessa técnica é levado em conta a origem de cada um dos casos de teste para garantir que ao menos um caso de teste seja gerado para cada requisito do sistema.

Resultados mostram que essas técnicas apresentam maior taxa de detecção de defeitos do que técnicas baseadas em cobertura por instruções ou por métodos (SINGH et al., 2012).

Outra forma de considerar os requisitos do sistema na priorização dos casos de testes são os trabalhos de Yoo e Harman (2012) e Stallbaum, Metzger e Pohl (2008). Esses trabalhos também são chamados de *risk-based approach*, uma vez que consideram o risco e a importância dos requisitos do sistema. Segundo essa abordagem, se um requisito é crítico e não pode falhar então os casos de teste vinculados a ele tem prioridade alta. Uma evolução desses trabalhos foi feita por Hettiarachchi, Do e Choi (2016), onde os autores utilizam um sistema de inferência *fuzzy* para melhorar o resultado da priorização.

3.2.3.7 Model-based approach

Esse tipo de abordagem utiliza informações de modelos UML (*Unified Modeling Language*) como o diagrama de sequência ou o diagrama de atividades para realizar a priorização dos casos de teste. Essa abordagem foi apresentada por Korel, Tahat e Harman (2005) que classificavam os casos de teste em um conjunto de alta prioridade e um conjunto de baixa prioridade de acordo a sua relevância para a modificação feita no modelo. Posteriormente, os casos de teste de cada grupo eram priorizados randomicamente. Korel, Koutsogiannakis e Tahat (2008) aprimoraram essa abordagem adicionando heurísticas baseadas na análise de dependência dos modelos. Uma dessas heurísticas defende que o caso de teste que exercita o maior número de transições, que é cada pequena alteração no código fonte que reflete no modelo, tem maior probabilidade de revelar defeitos, portanto deve ter maior prioridade. Nesse estudo, os autores afirmam que a abordagem baseada em modelo tem grande potencial por ser menos complexa que a abordagem baseada em cobertura.

3.2.3.8 Cost-aware approach

Essa abordagem considera que os casos de teste não têm o mesmo custo, ou seja, alguns podem ser mais custosos para executar do que outros. Dessa forma, essa abordagem defende que nem sempre é possível executar todos os casos de teste, então são priorizados os casos de teste até que se alcance um custo máximo

que seja factível de executar. Pode-se considerar que essa abordagem pode ser um complemento para as anteriores. Vários autores propuseram abordagens de priorização que se preocupam com o custo (*cost-aware*): Alspaugh et al. (2007), Wang et al. (2011), Parashar, Kalia e Bhatia (2012).

A grande diferença nessa abordagem está na métrica utilizada para medir a efetividade, uma vez que a métrica APFD considera que todas as defeitos têm a mesma relevância e que todos os casos de teste têm o mesmo custo. Por isso, Elbaum, Malishevsky e Rothermel (2001) estenderam a métrica APFD para considerar esses fatores, criando a $APFD_c$.

3.2.3.9 Resumo

Nesta Seção 3.2.3 foi apresentado o resultado do mapeamento sistemático conduzido sobre priorização de casos de testes no contexto de testes de regressão. Foram identificados oito grandes grupos de técnicas, as quais foram estudadas e detalhadas, mostrando a evolução das mesmas. Também foram identificadas as principais ferramentas envolvidas com a priorização de casos de teste, concluindo-se que não existe nenhuma ferramenta amplamente utilizada tanto pela academia quanto pela indústria. Com relação à métrica utilizada para medir a eficácia das propostas, a mais utilizada é a APFD já descrita na Seção 2.4. O mapeamento também identificou que somente 9,4% dos estudos encontrados realizaram estudo de caso real para validação das abordagens.

3.3 Ferramentas COCAR e Jira

Conforme visto anteriormente, o mapeamento sistemático conduzido não identificou nenhuma ferramenta consolidada e de ampla utilização, tanto pela academia como por empresas. As ferramentas citadas em geral tinham por objetivo validar as propostas dos autores. Nesse contexto, a ferramenta COCAR foi desenvolvida para dar suporte a algumas etapas do desenvolvimento de software, sobretudo no gerenciamento de requisitos e com aspectos de rastreabilidade. Isso permitiu que as propostas desenvolvidas por Di Thommazo (2014a) fossem avaliadas por meio de estudos experimentais. Apesar disso, não foram feitas

suficientes validações em sistemas reais. Isso ocorreu dada a dificuldade de uma empresa adequar seu processo de desenvolvimento com a ferramenta COCAR. Como já mencionado anteriormente, para minimizar essa dificuldade, foi desenvolvido um *plug-in* que dá suporte à abordagem de priorização em uma ferramenta comercial. Apesar do autor deste trabalho já ter experiência com o desenvolvimento da ferramenta COCAR, o desafio do desenvolvimento deste módulo em uma ferramenta comercial foi encarado como uma grande oportunidade para a condução de estudos experimentais em projetos de software reais. Na sequência é feita uma breve descrição da ferramenta COCAR e da ferramenta Jira (2018) que foi a escolhida como ferramenta comercial.

3.3.1 COCAR

A COCAR é uma ferramenta CASE planejada e desenvolvida por Di Thommazo (2007) e Martins (2007) que tem por objetivo dar suporte a algumas atividades do processo de desenvolvimento de software. Inicialmente ela foi desenvolvida para ambiente web, mas para tornar mais fácil a realização de estudos experimentais, ela foi migrada para ambiente desktop, com a utilização da linguagem Java. Dentre as principais funcionalidades da ferramenta COCAR, é possível destacar o gerenciamento de requisitos e as diferentes abordagens para geração da matriz de rastreabilidade de requisitos (RTM): (1) baseada em dados de entrada; (2) baseada em processamento de linguagem natural; (3) baseada em lógica *fuzzy*; (4) baseada em redes neurais artificiais.

Além dessas funcionalidades, a ferramenta também possui diversos módulos para dar suporte à condução de estudos experimentais. Esses módulos foram desenvolvidos para validar as abordagens de geração de matriz de rastreabilidade de requisitos do trabalho de Di Thommazo (2014a).

3.3.2 Jira

O Jira é uma ferramenta robusta que cobre todas as etapas do ciclo de vida de desenvolvimento de sistemas. Ela é utilizada por mais de 11500 organizações, e possui ampla documentação e suporte profissional.

As principais características do Jira são:

- Interface customizável;
- Campos e fluxos editáveis;
- Disponibilização de API para comunicação (REST/SOAP);
- Independência de plataforma (Windows, Linux e outros);
- Independência de banco de dados (PostgreSQL, Oracle, MySQL, SQLServer e outros);
- Alta escalabilidade;
- Esquemas de segurança e controle de acesso;
- Suporte a internacionalização;
- Integração com ambientes de desenvolvimento (Eclipse, Netbeans);
- Diversas opções de relatórios e *dashboards*.

Dentre as características citadas anteriormente, as que foram decisivas para a escolha desta ferramenta para este projeto de mestrado foram as três primeiras. Elas permitem que as funcionalidades disponíveis sejam personalizáveis para incorporar a abordagem proposta por este trabalho. A ferramenta Jira possui uma grande quantidade de *plug-ins* que podem ser desenvolvidos e colocados no ambiente das empresas. Atualmente ela dá suporte somente à abordagem *human-based*, onde a equipe de teste determina a prioridade de cada um dos casos de teste. Também é possível agrupar casos de teste que cada analista julgue importante para testar determinada funcionalidade.

Outro fator importante para a escolha da ferramenta Jira foi a existência de empresas parceiras do LaPES que utilizam essa ferramenta em seu processo de desenvolvimento. A escolha dessa ferramenta deu início a dois projetos de iniciação científica de alunos que estão migrando as abordagens de rastreabilidade hoje disponíveis somente na ferramenta COCAR para um *plug-in* de rastreabilidade que será disponibilizado na ferramenta Jira.

3.4 Considerações Finais

Esse capítulo apresentou a revisão bibliográfica elaborada para este trabalho, planejada com o paradigma GQM e realizada por meio de um mapeamento sistemático, envolvendo a priorização de casos de teste. Foram respondidas as questões levantadas, identificando quais são as principais técnicas de priorização, como os estudos são validados e quais são as principais ferramentas de apoio computacional. Também foram abordadas as duas ferramentas relacionadas a este trabalho, a ferramenta COCAR e a ferramenta Jira.

No próximo capítulo será apresentada a abordagem de priorização de casos de teste elaborada neste trabalho e o *plug-in* desenvolvido para a ferramenta Jira.

Capítulo 4

ABORDAGEM DE PRIORIZAÇÃO

Este capítulo apresenta a abordagem de priorização de casos de teste elaborada para este trabalho e o plug-in desenvolvido para a ferramenta Jira.

4.1 Considerações Iniciais

Como apresentado na Seção 1.2, o objetivo deste trabalho é propor uma solução para a aplicação da estratégia de testes de regressão – solução esta que integra a abordagem de priorização de casos de teste e o *plug-in* desenvolvido para o Jira. Portanto, neste capítulo, será explicado como a abordagem funciona e quais as diferenças para a abordagem de seleção de Di Thommazo (2015). Também será mostrado o *plug-in* desenvolvido e uma explicação de suas principais funcionalidades.

Este capítulo contém quatro seções. A primeira seção apresentou as considerações iniciais. A segunda seção descreve a abordagem de priorização de casos de teste, enquanto a terceira seção mostra o *plug-in* desenvolvido e suas funcionalidades. Por fim, a quarta seção faz as considerações finais.

4.2 Descrição da Abordagem de Priorização

A abordagem de priorização de casos de teste deste trabalho, que é baseada em rastreabilidade, foi elaborada para priorizar um conjunto de casos de teste de

regressão a partir de vários requisitos de uma vez. Essa é uma evolução em relação à abordagem de Di Thommazo (2015), que selecionava os casos de teste a partir de um requisito apenas. Essa mudança visa deixar a abordagem proposta mais próxima da realidade das empresas de software, que em sua maioria faz *sprints* de desenvolvimento. Dessa forma, ao final da *sprint*, elas têm a possibilidade de executar uma lista priorizada de casos de teste para testar tudo o que foi implementado dentro da *sprint*.

Como dito anteriormente, esta abordagem de priorização faz uso da Matriz de Rastreabilidade de Requisitos (RTM-E) proposta por Di Thommazo (2014a) para obter a dependência entre os requisitos, porém, diferente de Di Thommazo (2014a), que classifica essa dependência calculada em forte, fraca ou inexistente para selecionar os casos de teste associados, esta abordagem considera o valor da dependência entre os requisitos para priorizar os casos de teste.

Para exemplificar a diferença apontada acima, considere os quatro requisitos funcionais RF_A, RF_B, RF_C e RF_D. Suponha que a dependência entre RF_A e RF_B seja de 80%, que a dependência entre RF_A e RF_C seja de 60% e que a dependência entre RF_A e RF_D seja de 20%. Pela abordagem de Di Thommazo (2014a), a dependência entre RF_A e RF_D seria classificada como uma dependência fraca e as outras duas seriam classificadas como dependências fortes. Por consequência, caso haja alguma alteração no RF_A, e então seja necessário priorizar a lista de casos de teste de regressão, os casos de teste de RF_B e RF_C teriam a mesma prioridade. Pela abordagem proposta neste trabalho, os casos de teste de RF_B teriam maior prioridade que os casos de teste de RF_C porque RF_A tem maior dependência com RF_B (80%) do que com RF_C (60%).

A Figura 4.1 ilustra esse cenário apresentado.

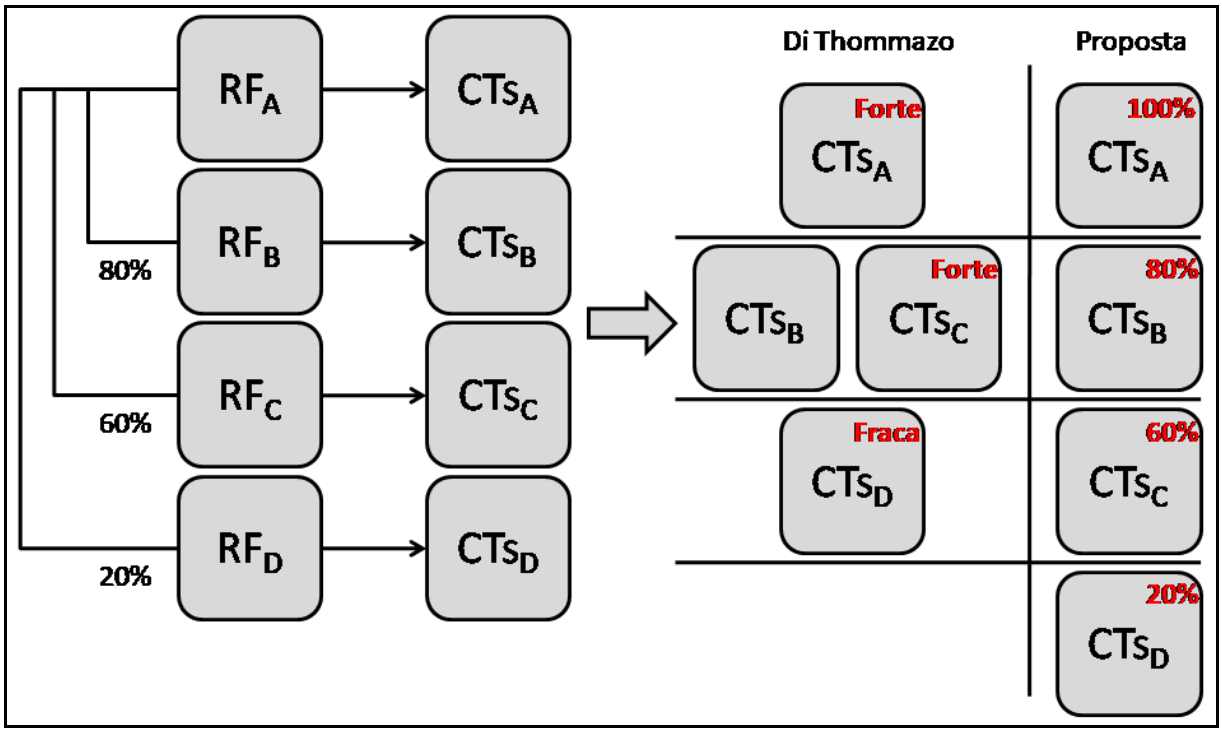


Figura 4.1 – Um cenário de priorização de casos de teste

Considerar o nível de dependência entre os requisitos para priorizar os casos de teste é mais assertivo que apenas considerar a classificação de dependência forte, fraca ou inexistente, pois como visto no exemplo acima, pode haver diferenças no nível de dependência entre requisitos, mas pela abordagem de Di Thommazo (2014a), essa diferença não é considerada.

Após essas importantes considerações acerca da abordagem proposta, abaixo será explicado o algoritmo utilizado para priorizar os casos de teste:

- Passo 1: Selecionar um requisito da *sprint*;
- Passo 2: Selecionar todos os casos de teste que pertencem àquele requisito. Adicionar esses casos de teste à lista de casos de teste de regressão com 100% de importância, pois são os casos de teste que estão diretamente ligados ao requisito em questão;
- Passo 3: Gerar a Matriz de Rastreabilidade de Requisitos (RTM-E);
- Passo 4: Selecionar todos os casos de teste pertencentes aos requisitos que apresentam alguma dependência com o requisito em questão. Adicionar tais casos de teste à lista de casos de teste de regressão com a importância equivalente à dependência que seus

respectivos requisitos têm com o requisito em questão. Repetir os quatro primeiros passos para cada requisito da *sprint*;

- Passo 5: Normalizar os valores de importância de todos os casos de teste que estão na lista de casos de teste de regressão;
- Passo 6: Priorizar a lista de casos de teste de regressão pelo valor de importância dos casos de teste.

O resultado final do algoritmo é uma lista de casos de teste de regressão, sem casos de teste repetidos, priorizada pela importância dos casos de teste dentro da *sprint*.

A normalização citada acima é um passo importante dentro do algoritmo de priorização. Como explicado, o algoritmo faz o processamento dos casos de teste associados, analisando requisito por requisito em uma *sprint*. Por isso, por questão de dependência entre requisitos, é comum que casos de teste sejam adicionados à lista de casos de teste de regressão mais de uma vez. No entanto, não faz sentido que o mesmo caso de teste seja executado mais de uma vez em momentos diferentes na aplicação dos testes de regressão. Então a solução adotada para que a lista de casos de teste de regressão não apresente casos de teste repetidos foi a de somar os valores de importância desses casos e normalizar todas as importâncias pela maior soma obtida.

É sabido que essa não é a única forma de resolver esse problema. Outra solução poderia considerar a quantidade de vezes que o caso de teste aparece e dar uma importância maior a esse caso de teste. Porém, este trabalho não explora diferentes formas de remover os casos de teste repetidos porque seriam necessários mais estudos experimentais e muito mais dados de sistemas. Sugere-se como trabalhos futuros explorar outras formas e avaliar qual seria a melhor delas.

A Figura 4.2 apresenta um fluxograma com os passos do algoritmo de priorização, conforme explicado acima.

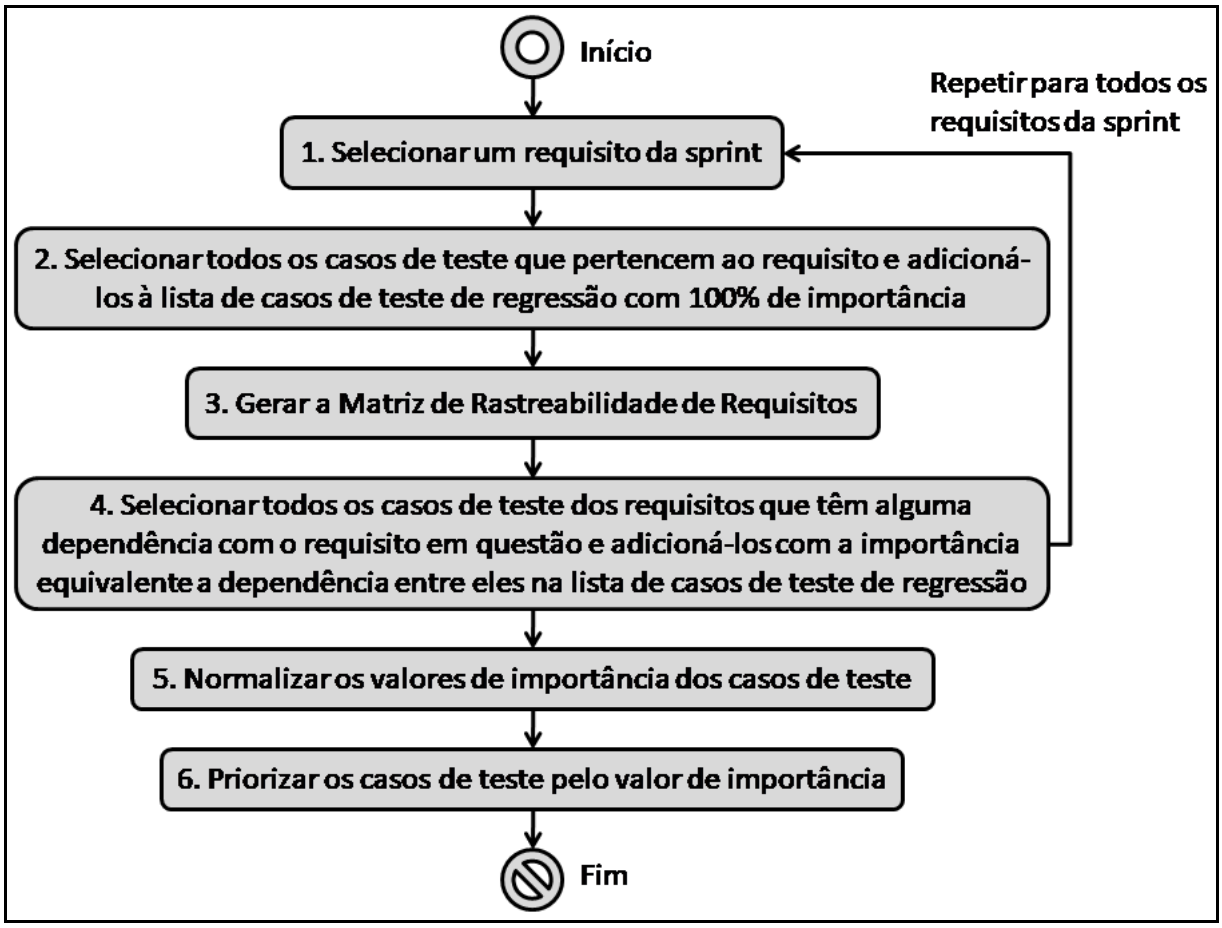


Figura 4.2 – Fluxograma do algoritmo de priorização

4.3 Recurso Automatizado para Abordagem de Priorização como Plug-in para o Jira

Conforme explicado anteriormente, a ideia de criar uma ferramenta que apoiasse a priorização de casos de teste surgiu durante o levantamento da bibliografia relacionada a este trabalho, no qual notou-se que poucos estudos da área apresentam uma ferramenta em conjunto com as abordagens propostas. Nesse sentido, a primeira iniciativa foi criar um módulo para a ferramenta COCAR, que já é uma ferramenta que incorpora as abordagens de Di Thommazo (2014a) para geração da Matriz de Rastreabilidade de Requisitos, entre outras funcionalidades para gerenciamento de requisitos e casos de teste. Porém, a maioria das empresas de software já tem suas ferramentas e processos bem definidos e se adaptar a uma nova ferramenta implica em uma série de novos desafios. Por isso, decidiu-se

disponibilizar o suporte à abordagem de priorização proposta em uma ferramenta que já fosse bastante utilizada pelas empresas. Então a solução encontrada foi a criação de um *plug-in* para o Jira.

O *plug-in* desenvolvido neste trabalho para a ferramenta Jira tem o formato de uma aplicação web. Essa aplicação foi hospedada em um servidor virtual na nuvem e está disponível em um endereço válido da internet. Seu desenvolvimento seguiu as orientações e requisitos da Atlassian – empresa que desenvolve e mantém o Jira, de forma que é possível disponibilizá-lo no *marketplace* do Jira para que outras pessoas e empresas utilizem.

A Figura 4.3 ilustra a arquitetura da aplicação web como *plug-in* para o Jira.

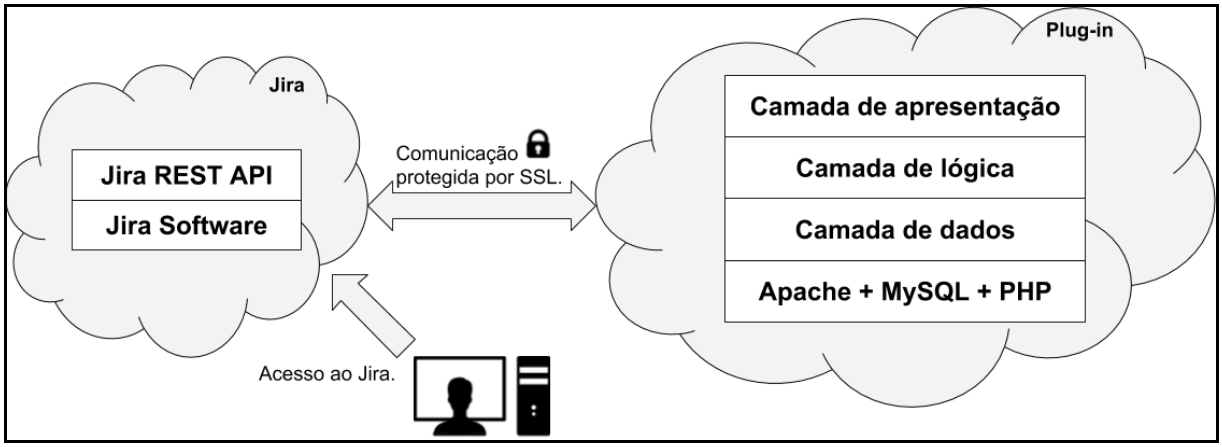


Figura 4.3 – Arquitetura do plug-in desenvolvido para o Jira

O *plug-in* está dividido em três camadas: camada de dados, que acessa e disponibiliza os dados armazenados no banco de dados; a camada de lógica, que é responsável por todas as regras e pela lógica da aplicação; camada de apresentação, que é a interface de interação com o usuário. A linguagem utilizada para o desenvolvimento foi PHP, mas poderia ter sido qualquer outra linguagem para internet porque isso não é um requisito do Jira. Como servidor de banco de dados foi utilizado o MySQL e como servidor web o Apache, mas assim como a linguagem, essas ferramentas não são obrigatórias, e poderiam ser substituídas por outras.

Como estão em servidores diferentes, toda comunicação entre o Jira e o *plug-in* ocorre pela internet, por isso o Jira exige que essa comunicação seja protegida por SSL (*Secure Socket Layer*), ou seja, toda essa comunicação é protegida por criptografia. Essa exigência de segurança na comunicação se faz necessária porque

o Jira acessa dados do *plug-in* para exibir ao usuário e o *plug-in* também acessa dados do Jira através de sua API REST para obter informações necessárias para seu funcionamento.

Abaixo são apresentadas as telas do *plug-in* e as suas funcionalidades.

A Figura 4.4 apresenta a tela inicial do *plug-in*, com um menu lateral com as funcionalidades disponíveis divididas em grupos.

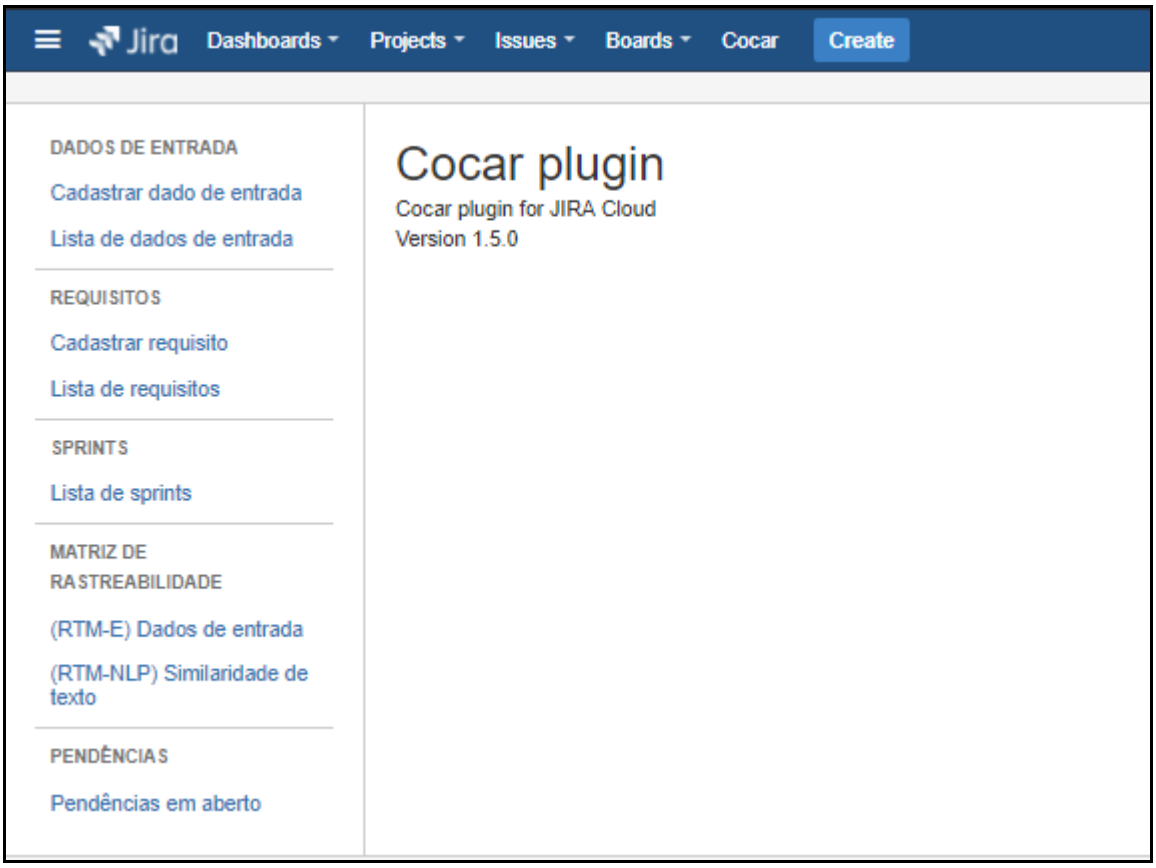


Figura 4.4 – Tela inicial do plug-in

O primeiro grupo agrega as funcionalidades de gerenciamento dos dados de entrada. É possível cadastrar, editar e visualizar dados de entrada de cada projeto. Para cadastrar um novo dado de entrada é necessário informar o nome do dado de entrada, uma descrição, uma restrição e qual o tipo do dado de entrada. A Figura 4.5 apresenta a tela de listagem dos dados de entrada. Ao clicar em algum dado de entrada da lista, os dados são exibidos no formulário à direita.

Figura 4.5 – Tela de listagem de dados de entrada

O segundo grupo agrega as funcionalidades de gerenciamento dos requisitos. Assim como os dados de entrada, também é possível cadastrar, editar e visualizar os requisitos de cada projeto. O cadastro de requisitos possui o nome do requisito, uma descrição, um processamento, restrições do requisito e a saída esperada. Além desses atributos, também é necessário informar quais dados de entrada estão relacionados com o requisito. Esse passo no cadastro dos requisitos é muito importante, porque são nessas informações que a abordagem se baseia para primeiramente determinar o nível de dependência dos requisitos e posteriormente priorizar os casos de teste.

A Figura 4.6 apresenta a tela de listagem dos requisitos. Como é possível notar, há um requisito selecionado, e suas informações estão sendo exibidas no formulário à direita.

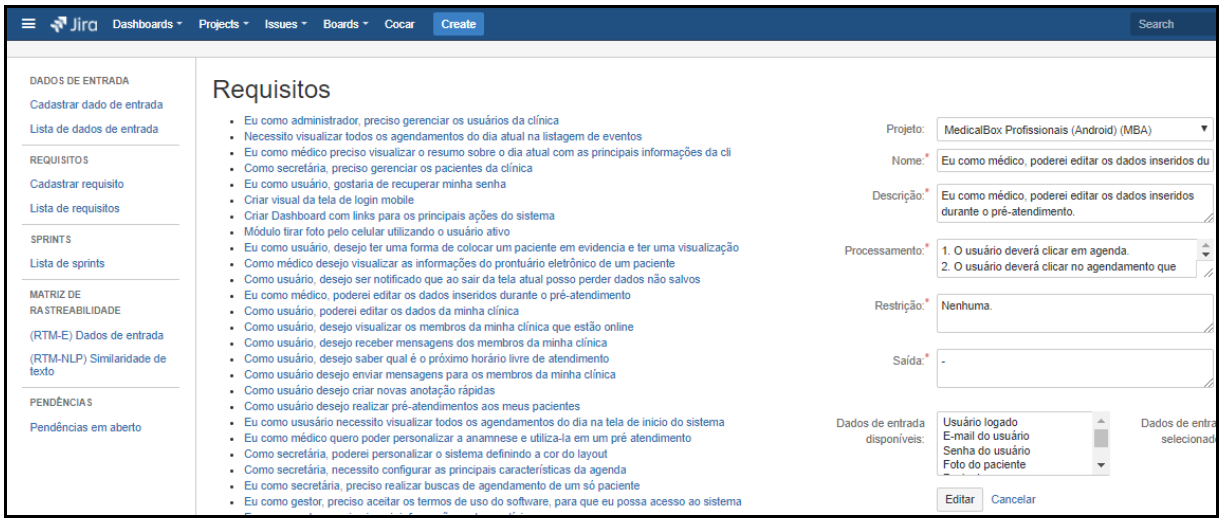


Figura 4.6 – Tela de listagem de requisitos

O terceiro grupo tem apenas uma funcionalidade, a de exibir as *sprints* de um projeto. Porém, é a partir dessa tela que o usuário pode gerar a lista de casos de teste priorizada. Para isso, basta selecionar uma *sprint* e clicar no botão “Gerar”. Então o *plug-in* se encarregará de fazer todo o processamento e exibir a lista priorizada de casos de teste para serem executados para esta *sprint*. A Figura 4.7 apresenta a tela de listagem das *sprints*.

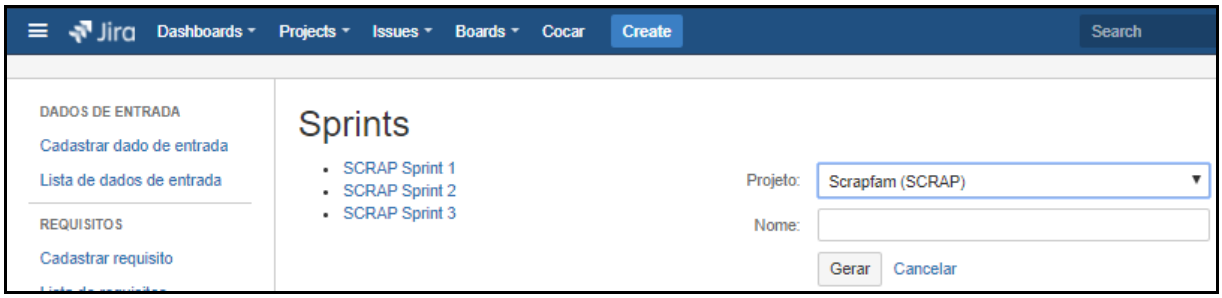
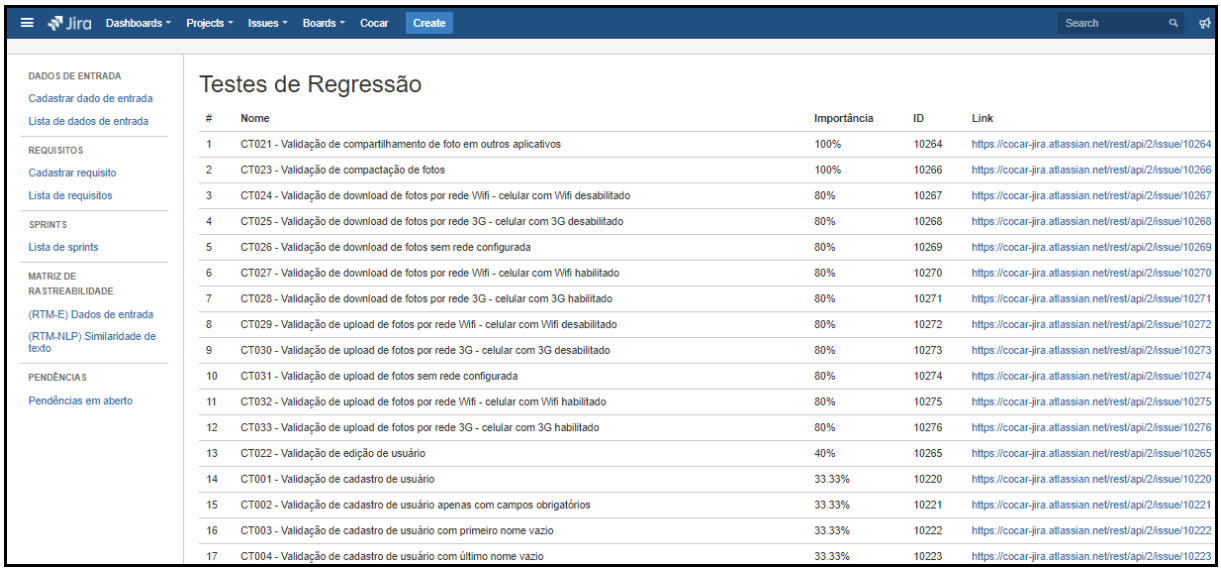


Figura 4.7 – Tela de listagem de sprints

Como explicado anteriormente, ao selecionar uma *sprint* e clicar no botão “Gerar” o *plug-in* irá fazer a priorização dos casos de teste de regressão e exibir uma lista em uma nova tela. Essa tela com a lista priorizada dos casos de teste de regressão é exibida na Figura 4.8.



#	Nome	Importância	ID	Link
1	CT021 - Validação de compartilhamento de foto em outros aplicativos	100%	10264	https://cocar-jira.atlassian.net/rest/api/2/issue/10264
2	CT023 - Validação de compactação de fotos	100%	10266	https://cocar-jira.atlassian.net/rest/api/2/issue/10266
3	CT024 - Validação de download de fotos por rede Wifi - celular com Wifi desabilitado	80%	10267	https://cocar-jira.atlassian.net/rest/api/2/issue/10267
4	CT025 - Validação de download de fotos por rede 3G - celular com 3G desabilitado	80%	10268	https://cocar-jira.atlassian.net/rest/api/2/issue/10268
5	CT026 - Validação de download de fotos sem rede configurada	80%	10269	https://cocar-jira.atlassian.net/rest/api/2/issue/10269
6	CT027 - Validação de download de fotos por rede Wifi - celular com Wifi habilitado	80%	10270	https://cocar-jira.atlassian.net/rest/api/2/issue/10270
7	CT028 - Validação de download de fotos por rede 3G - celular com 3G habilitado	80%	10271	https://cocar-jira.atlassian.net/rest/api/2/issue/10271
8	CT029 - Validação de upload de fotos por rede Wifi - celular com Wifi desabilitado	80%	10272	https://cocar-jira.atlassian.net/rest/api/2/issue/10272
9	CT030 - Validação de upload de fotos por rede 3G - celular com 3G desabilitado	80%	10273	https://cocar-jira.atlassian.net/rest/api/2/issue/10273
10	CT031 - Validação de upload de fotos sem rede configurada	80%	10274	https://cocar-jira.atlassian.net/rest/api/2/issue/10274
11	CT032 - Validação de upload de fotos por rede Wifi - celular com Wifi habilitado	80%	10275	https://cocar-jira.atlassian.net/rest/api/2/issue/10275
12	CT033 - Validação de upload de fotos por rede 3G - celular com 3G habilitado	80%	10276	https://cocar-jira.atlassian.net/rest/api/2/issue/10276
13	CT022 - Validação de edição de usuário	40%	10265	https://cocar-jira.atlassian.net/rest/api/2/issue/10265
14	CT001 - Validação de cadastro de usuário	33.33%	10220	https://cocar-jira.atlassian.net/rest/api/2/issue/10220
15	CT002 - Validação de cadastro de usuário apenas com campos obrigatórios	33.33%	10221	https://cocar-jira.atlassian.net/rest/api/2/issue/10221
16	CT003 - Validação de cadastro de usuário com primeiro nome vazio	33.33%	10222	https://cocar-jira.atlassian.net/rest/api/2/issue/10222
17	CT004 - Validação de cadastro de usuário com último nome vazio	33.33%	10223	https://cocar-jira.atlassian.net/rest/api/2/issue/10223

Figura 4.8 – Tela com a lista de casos de teste de regressão

Por fim, no quarto grupo, há a funcionalidade de gerar a Matriz de Rastreabilidade de Requisitos baseada em dados de entrada (RTM-E). Ela é uma matriz identidade que mostra a dependência de cada requisito com todos os outros requisitos do mesmo projeto. Essa funcionalidade é usada basicamente para visualização e conferência dos dados. A Figura 4.9 apresenta a tela com a Matriz de Rastreabilidade de Requisitos (RTM-E).

-	RF002 - Login do usuário	RF003 - Recuperar senha	RF004 - Tirar foto	RF005 - Adicionar amigo	RF006 - Cadastro de tag	RF007 - Compartilhar tag no app	RF008 - Compartilhar foto no app	RF009 - Compartilhar tag externamente	RF010 - Compartilhar foto externamente	RF001 - Cadastro de usuário	RF011 - Edição de usuário	RF012 - Sincronização de fotos
RF002 - Login do usuário	-	50	0	50	0	0	0	0	0	33.333333333333	16.666666666667	0
RF003 - Recuperar senha	50	-	0	100	0	0	0	0	0	16.666666666667	20	0
RF004 - Tirar foto	0	0	-	0	0	0	0	0	0	0	0	0
RF005 - Adicionar amigo	50	100	0	-	0	0	0	0	0	16.666666666667	20	0
RF006 - Cadastro de tag	0	0	0	0	-	0	0	0	0	0	0	0

Figura 4.9 – Tela da Matriz de Rastreabilidade de Requisitos (RTM-E)

As funcionalidades que aparecem no menu do *plug-in* mas que não foram explicadas nesta seção não estão relacionadas ao contexto deste trabalho.

4.4 Considerações Finais

Este capítulo apresentou a abordagem de priorização de casos de teste e o *plug-in* desenvolvido para o Jira, descreveu como é o algoritmo de priorização e as funcionalidades do *plug-in*. Essas são as principais contribuições deste trabalho para os problemas da área de desenvolvimento de software contextualizados anteriormente.

O próximo capítulo mostra os estudos experimentais conduzidos com o intuito de avaliar a abordagem de priorização e o *plug-in*, assim como os resultados obtidos.

Capítulo 5

ESTUDOS EXPERIMENTAIS

Este capítulo apresenta os estudos experimentais conduzidos durante este trabalho de mestrado e os resultados obtidos.

5.1 Considerações Iniciais

Como citado anteriormente na metodologia de trabalho adotada, estudos experimentais seriam conduzidos para avaliar a abordagem de priorização proposta e o *plug-in* desenvolvido para o Jira. Dessa forma, dois estudos foram conduzidos com sistemas reais. Para cada um desses estudos será apresentado o contexto em que ele se insere, bem como seu objetivo, os participantes, os artefatos, a métrica, os resultados, as análises desses resultados e as ameaças à validade.

O primeiro estudo acompanhou o início do desenvolvimento do Scrapfam, um sistema móvel de compartilhamento de mídias digitais. Os resultados obtidos foram satisfatórios, mas decidiu-se realizar um novo estudo para comprovar a eficácia da abordagem. O segundo estudo foi realizado em parceria com a empresa Monitora, que forneceu os dados de um sistema médico, denominado Medical Box. Este é um sistema bem maior que o utilizado no primeiro estudo, já tem versões em produção e ainda possui funcionalidades sendo desenvolvidas.

Em ambos os estudos de caso utilizou-se a métrica *Average Percentage of Faults Detected* (APFD) para medir a eficácia da abordagem de priorização. Para o segundo estudo, foram feitas comparações dos valores obtidos pela métrica APFD da abordagem de priorização proposta com outras abordagens: priorização randômica, utilizada como base de comparação por grande parte dos trabalhos da

área; priorização manual, utilizada pela equipe de garantia de qualidade ao executar os casos de teste; priorização ideal, que também pode ser considerada como um gabarito.

Também foram feitas comparações dos resultados obtidos pelo segundo estudo de caso com outras abordagens de priorização de casos de teste encontradas na literatura. Os estudos de Zhang et al. (2009), Carlson, Do e Denton (2011), Rajarathinam e Natarajan (2013), Indumathi e Selvamani (2015) foram selecionados para comparação.

Este capítulo contém cinco seções. A primeira seção fez as considerações iniciais do capítulo. A segunda seção apresenta o estudo experimental realizado com o sistema Scrapfam, enquanto a terceira seção apresenta outro estudo experimental, realizado no sistema Medical Box da empresa Monitora. A quarta seção mostra algumas comparações dos resultados obtidos pela abordagem proposta neste trabalho com outras abordagens encontradas na literatura. Por fim, a quinta seção faz as considerações finais.

5.2 Estudo de Caso I

Contexto

Este estudo experimental foi realizado com uma *start-up* que idealizou o Scrapfam, um sistema móvel de compartilhamento de mídias digitais. O sistema permite que os usuários tirem fotos, gravem áudios e vídeos e compartilhem com outros usuários do sistema. Também é possível criar rótulos, denominados *tags*, para classificar e organizar os conteúdos.

O estudo de caso acompanhou o Scrapfam desde as primeiras etapas do processo de desenvolvimento. Utilizou-se um processo iterativo e incremental, no qual em cada *sprint*, correções e melhorias eram feitas e também novas funcionalidades eram entregues. O Scrapfam não possui muitos requisitos e foi escolhido justamente por esse motivo para ser o estudo inicial deste trabalho, porque dessa forma, pôde ser conduzido com mais controle. Além disso, este estudo de caso permitiu que alguns defeitos no *plug-in* fossem encontrados e corrigidos.

Objetivo

Avaliar a eficácia da abordagem de priorização proposta, e também analisar a utilização do *plug-in* desenvolvido, na ferramenta Jira.

Participantes

Os participantes do estudo de caso foram os três fundadores da *start-up*, que atuaram como desenvolvedores e uma estudante de iniciação científica, que auxiliou em várias atividades durante o estudo de caso.

Artefatos

Requisitos

- Elaborados pelos desenvolvedores;
- Cadastrados no Jira através do *plug-in* desenvolvido.

Casos de teste

- Elaborados pelo autor deste trabalho e pela testadora;
- Cadastrados diretamente no Jira.

Lista priorizada de casos de teste

- Gerada pelo *plug-in* de acordo com a abordagem proposta.

Métrica

A métrica utilizada foi a *Average Percentage of Faults Detected* (APFD). Conforme explicada na Seção 2.4, essa métrica é utilizada para medir a taxa de detecção de defeitos de um conjunto priorizado de casos de teste. Em resumo, quanto maior a taxa de detecção de defeitos, mais rápida aquela sequência de casos de teste é capaz de revelar os defeitos.

Execução

Antes da execução de fato, o estudo de caso foi planejado e dividido em alguns passos. Esses passos são descritos abaixo:

- Passo 1: Cadastramento dos requisitos do sistema por meio do *plug-in* desenvolvido. Passo realizado pelo autor deste trabalho e pela aluna de iniciação científica;
- Passo 2: Cadastramento dos casos de teste diretamente no Jira. Passo realizado pelo autor deste trabalho e pela aluna de iniciação científica;
- Passo 3: Criação de uma *sprint* no Jira e definição de quais requisitos entram na *sprint* para serem desenvolvidos. Passo realizado pelo autor deste trabalho em conjunto com os desenvolvedores;
- Passo 4: Desenvolvimento das atividades que foram definidas para a *sprint*. Essas atividades foram divididas entre os dois desenvolvedores;
- Passo 5: Utilização do *plug-in* desenvolvido para obtenção da lista priorizada de casos de teste da *sprint*. Passo realizado pelo autor deste trabalho;
- Passo 6: Execução dos casos de teste de acordo com a priorização definida no passo anterior. Todos os defeitos encontrados foram cadastrados no Jira a fim de manter o registro e o histórico de tudo o que ocorreu durante a *sprint*. Essas atividades foram divididas entre o autor deste trabalho e a aluna de iniciação científica.

É importante salientar que os passos relatados acima não ocorreram apenas uma vez durante todo o estudo experimental. Por ter sido adotado um processo de desenvolvimento iterativo e incremental, como dito anteriormente, os passos 3, 4, 5 e 6 se repetiram a cada *sprint*. Apenas os passos 1 e 2 não se repetiram porque foram cadastrados inicialmente dez requisitos que foram sendo distribuídos ao longo de algumas *sprints*.

Abaixo são detalhadas as *sprints* de desenvolvimento, mostrando quais requisitos foram implementados e qual a sequência de casos de teste executada.

Sprint 1

A primeira *sprint* de desenvolvimento contou com seis requisitos dos dez que foram inicialmente cadastrados. Os requisitos escolhidos foram:

- RF001 - Cadastro de usuário
- RF002 - Login do usuário
- RF004 - Tirar foto
- RF005 - Adicionar amigo
- RF006 - Cadastro de tag
- RF007 - Compartilhar tag no app

Para esses seis requisitos funcionais foram elaborados vinte casos de teste. Após a execução dos casos de teste, seguindo a priorização determinada pela abordagem proposta, foram encontrados oito defeitos.

A Tabela 5.1 apresenta um resumo da *Sprint* 1. É possível observar a priorização dos casos de teste feita pela abordagem proposta e o respectivo nível de dependência normalizado para cada caso de teste. Também pode-se visualizar qual caso de teste detectou determinado defeito.

Tabela 5.1 – Lista priorizada de casos de teste e defeitos encontrados na Sprint 1 (Estudo de Caso I).

#	Caso de teste	Importância	Defeitos
1	CT010	100%	
2	CT011	100%	
3	CT012	100%	
4	CT014	89,47%	
5	CT015	89,47%	
6	CT016	89,47%	BUG006
7	CT001	84,21%	BUG001
8	CT002	84,21%	
9	CT003	84,21%	
10	CT004	84,21%	BUG002
11	CT005	84,21%	
12	CT006	84,21%	
13	CT007	84,21%	
14	CT008	84,21%	BUG003
15	CT009	84,21%	BUG004
16	CT013	52,63%	BUG005
17	CT017	52,63%	
18	CT018	52,63%	
19	CT019	52,63%	BUG007
20	CT020	52,63%	BUG008

Sprint 2

Na segunda *sprint* não entraram novos requisitos para serem desenvolvidos. A equipe de desenvolvimento tomou a decisão de aprimorar e corrigir requisitos da primeira *sprint*. Um dos requisitos também foi atualizado. Dessa forma, a *sprint* contou com cinco requisitos, sendo eles:

- RF001 - Cadastro de usuário
- RF004 - Tirar foto
- RF005 - Adicionar amigo
- RF006 - Cadastro de tag
- RF007 - Compartilhar tag no app

Para esses cinco requisitos funcionais foram utilizados os mesmos casos de teste da *sprint* anterior, ou seja, não foi necessária a elaboração de nenhum caso de teste novo. Porém, com a alteração de um requisito e a retirada de outro, era esperado que a priorização dos casos de teste para esta *sprint* fosse diferente. Então, após a execução dos casos de teste foram encontrados seis defeitos, sendo que três defeitos encontrados na *Sprint* 1 foram corrigidos e um novo defeito foi encontrado. A Tabela 5.2 apresenta o resumo da *Sprint* 2:

Tabela 5.2 – Lista priorizada de casos de teste e defeitos encontrados na Sprint 2 (Estudo de Caso I).

#	Caso de teste	Importância	Defeitos
1	CT001	100%	
2	CT002	100%	BUG009
3	CT003	100%	
4	CT004	100%	
5	CT005	100%	
6	CT006	100%	
7	CT007	100%	
8	CT008	100%	BUG003
9	CT009	100%	BUG004
10	CT014	100%	
11	CT015	100%	
12	CT016	100%	BUG006
13	CT013	85,71%	BUG005
14	CT017	85,71%	
15	CT018	85,71%	
16	CT019	85,71%	BUG007
17	CT020	85,71%	
18	CT010	71,43%	
19	CT011	71,43%	
20	CT012	71,43%	

Sprint 3

Na terceira *sprint* foram desenvolvidos quatro requisitos, entre eles um requisito que havia sido cadastrado previamente e três novos requisitos. Esses três novos requisitos foram adicionados porque a equipe do projeto julgou que eram

importantes para o funcionamento do aplicativo. Os requisitos da terceira *sprint* foram:

- RF010 - Compartilhar foto externamente
- RF011 - Edição de usuário
- RF012 - Sincronização de fotos
- RF013 - Compactação de fotos

Também foram criados treze novos casos de teste para esses quatro requisitos, totalizando 33 casos de teste para todos os requisitos desenvolvidos. Após a priorização dos casos de teste, 28 deles foram executados e seis defeitos foram encontrados. A Tabela 5.3 apresenta o resumo da *Sprint* 3:

Tabela 5.3 – Lista priorizada de casos de teste e defeitos encontrados na Sprint 3 (Estudo de Caso I).

#	Caso de teste	Importância	Defeitos
1	CT021	100%	
2	CT023	100%	
3	CT024	80%	BUG010
4	CT025	80%	BUG011
5	CT026	80%	BUG012
6	CT027	80%	
7	CT028	80%	
8	CT029	80%	
9	CT030	80%	
10	CT031	80%	
11	CT032	80%	
12	CT033	80%	
13	CT022	40%	
14	CT001	33,33%	
15	CT002	33,33%	
16	CT003	33,33%	
17	CT004	33,33%	
18	CT005	33,33%	
19	CT006	33,33%	
20	CT007	33,33%	
21	CT008	33,33%	BUG003
22	CT009	33,33%	BUG004
23	CT014	8%	
24	CT015	8%	
25	CT016	8%	BUG006
26	CT010	6,67%	
27	CT011	6,67%	
28	CT012	6,67%	

Resultados

Os resultados calculados com a métrica APFD de cada *sprint* são apresentados na Tabela 5.4.

Tabela 5.4 – Resultados do Estudo de Caso I.

	APFD Calculado
***Sprint* 1**	0,35625
***Sprint* 2**	0,525
***Sprint* 3**	0,54167

Análise dos Resultados

Os resultados obtidos nesse estudo de caso, ainda que tenha sido feito em um sistema pequeno, foram satisfatórios, levando em consideração outras abordagens de priorização observadas na literatura. Mesmo assim, viu-se a necessidade de realizar outro estudo de caso com um sistema bem maior, para verificar se realmente a abordagem apresenta boa eficácia, de forma que ela contribua na prática com a priorização de casos de teste. Este outro estudo de caso maior é apresentado na seção seguinte.

Ameaças à Validade

Este estudo possui algumas ameaças à validade. É possível mencionar o tamanho do sistema utilizado, que possui poucos requisitos e poderia não retratar a realidade de grandes empresas de software que atuam no mercado. Outra ameaça seria em relação aos casos de teste elaborados, que podem não ter encontrado certos defeitos, o que causaria mudança nos valores obtidos de APFD, podendo ser tanto para cima quanto para baixo, dependendo da priorização.

5.3 Estudo de Caso II

Contexto

Este estudo de caso foi realizado em parceria com a Monitora, que é uma empresa de desenvolvimento de software instalada em São Carlos há 7 anos e que conta atualmente com mais de 100 funcionários. Um dos sistemas de seu portfólio é o Medical Box, que foi selecionado para este estudo. O Medical Box é um sistema

da área médica, que conta com 73 requisitos e mais de 100 casos de teste (até o momento da realização do estudo). Esse sistema vem sendo desenvolvido há mais de dois anos e ele conta com uma equipe exclusiva de desenvolvedores e analistas de qualidade. O Medical Box já está em produção e ainda tem novas funcionalidades sendo desenvolvidas.

Dentre as principais funcionalidades do Medical Box pode-se destacar o gerenciamento de clínicas, médicos, funcionários e pacientes. Também é responsável pelo agendamento de consultas e atendimentos.

Objetivo

Avaliar a eficácia da abordagem de priorização proposta.

Participantes

Os participantes do estudo de caso foram o autor deste trabalho, a gerente de qualidade da empresa Monitora e o responsável pelo projeto Medical Box.

Artefatos

Arquivo contendo todos os dados do projeto Medical Box

- Exportado do Jira da Monitora;
- Contém os requisitos e os casos de teste do projeto, que foram usados na realização do estudo de caso;
- Importado no Jira utilizado para a condução dos estudos experimentais deste trabalho.

Lista priorizada de casos de teste

- Gerada pelo *plug-in* de acordo com a abordagem proposta.

Métrica

Assim como no estudo de caso anterior, a métrica utilizada foi a *Average Percentage of Faults Detected* (APFD).

Para este estudo de caso, além de calcular o APFD da priorização obtida pelo *plug-in*, como foi feito no primeiro estudo de caso, também foram calculados valores de APFD obtidos por outras abordagens de priorização. Dessa forma, foi possível comparar a eficácia da abordagem proposta, a cada *sprint*, em relação a outras abordagens. A primeira abordagem utilizada para comparação foi a priorização randômica. Essa abordagem é muito utilizada por trabalhos da área para comparação de resultados, portanto decidiu-se utilizar neste trabalho também. A segunda abordagem de comparação foi a priorização manual, considerada como a priorização que a equipe de garantia de qualidade da Monitora utilizou para executar os testes da *sprint*. E a terceira abordagem foi a priorização ideal, que também pode ser chamada de gabarito, porque é a priorização que obtém o maior valor de APFD possível da *sprint*.

Portanto, em cada *sprint* será feito esse comparativo entre a abordagem proposta e as abordagens randômica, manual e ideal. É esperado que a abordagem proposta apresente maior eficácia do que as abordagens randômica e principalmente a manual, pois seria uma grande contribuição, uma vez que o *plug-in* poderia ser usado para substituir a decisão humana ao priorizar os casos de teste. Em relação à priorização ideal, espera-se que abordagem proposta chegue o mais próximo possível, sendo que o cenário perfeito seria quando a abordagem proposta priorizasse os casos de teste com a mesma sequência da priorização ideal.

Execução

Para a realização deste estudo de caso, a Monitora disponibilizou todos os dados do Medical Box. Esses dados estavam cadastrados no Jira utilizado por eles e foram exportados para outro Jira utilizado na realização dos estudos deste trabalho. Após a importação de todos os dados do sistema, foi necessário apenas mais um passo para dar início ao estudo de caso. Como os dados dos requisitos já estavam cadastrados no formato que a Monitora costuma utilizar, foi preciso extrair deles os seus dados de entrada, para serem cadastrados pelo *plug-in* desenvolvido. Como explicado na Seção 4.2, o *plug-in* utiliza os dados de entrada dos requisitos para calcular a dependência entre eles e gerar a Matriz de Rastreabilidade de Requisitos, e posteriormente utiliza essa dependência para priorizar os casos de teste. A extração dos dados de entrada dos requisitos foi feita pelo autor deste trabalho de

forma manual, analisando a descrição e o processamento de cada um. Os dados de entrada extraídos foram validados com o responsável pelo projeto.

A condução deste estudo de caso considerou 3 *sprints* de desenvolvimento do Medical Box. Essas *sprints* não são, de fato, as três primeiras *sprints* de desenvolvimento do sistema, mas para facilitar o entendimento, são chamadas de *Sprints* 1, 2 e 3. Elas são apresentadas com detalhes a seguir. Para cada uma delas é mostrado quais requisitos foram implementados e qual a sequência de casos de teste executada.

Sprint 1

A primeira *sprint* contou com 10 requisitos. Entre eles, são contempladas funcionalidades de visualização de informações de prontuário do paciente, envio e recebimento de mensagens para outros membros da clínica, cadastro de informações da clínica, buscas de agendamentos de pacientes, entre outras. Para esses requisitos, o *plug-in* priorizou 9 casos de teste, e após a execução deles, foram encontrados 11 defeitos. A Tabela 5.5 apresenta o resumo das informações obtidas da *Sprint* 1.

Tabela 5.5 – Lista priorizada de casos de teste e defeitos encontrados na Sprint 1 (Estudo de Caso II).

#	Caso de teste	Importância	Defeitos
1	User Journey - Paciente em evidência	100%	MBA-682
2	Visualização atendimentos	50%	MBA-673
3	[QA] Agendamento	37.5%	MBA-818, MBA-819, MBA-833, MBA-870
4	[QA] Agendamento - Recorrência	25%	MBA-918, MBA-932
5	[DEV] - Agendamento - Recorrências (MBA-899)	25%	
6	[DEV] Performance e Estabilidade no Sistema	16.67%	
7	[QA] Performance e Estabilidade no Sistema	16.67%	MBA-928
8	[QA] Chat	16.67%	MBA-793
9	Visualização dos agendamentos do dia	16.67%	MBA-820

Sprint 2

A segunda *sprint* contou com 10 requisitos também. Entre eles, são contempladas funcionalidades de *login*, recuperação de senha, cadastro de usuário, criação de anotações, visualização de pacientes, entre outras. Para esses requisitos, o *plug-in* priorizou 12 casos de teste, e após a execução deles, foram encontrados 13 defeitos. A Tabela 5.6 apresenta o resumo das informações obtidas da *Sprint* 2.

Tabela 5.6 – Lista priorizada de casos de teste e defeitos encontrados na Sprint 2 (Estudo de Caso II).

#	Caso de teste	Importância	Defeitos
1	[QA] Login	100%	MBA-829, MBA-846
2	Criar nova conta	92.31%	
3	[DEV] Prontuário - Adição de informações pós-atendimento (MBA-942)	46.15%	
4	[DEV] Navegação - Sistema (MBA-851)	46.15%	MBA-663, MBA-730, MBA-851
5	[QA] Atendimento - Conexão Wifi	46.15%	MBA-769, MBA-771, MBA-772, MBA-921
6	[QA] Navegação - Sistema	46.15%	MBA-839, MBA-845
7	[QA] Chat	46.15%	MBA-808
8	Persistência de dados	46.15%	MBA-757
9	[QA]Cadastro de paciente - Dados Gerais	46.15%	
10	[DEV] Anexo de Imagem - Processamento da Imagem (MBA-1036)	34.62%	
11	[QA] Prontuário	34.62%	
12	[QA] Pagamento - Adição de Usuários no Sistema	30.77%	

Sprint 3

A terceira *sprint* contou com 6 requisitos. Entre eles, são contempladas funcionalidades de cadastro de procedimentos, notificação de recebimento de ligações, agendamentos recorrentes, entre outras. Para esses requisitos, o *plug-in* priorizou 8 casos de teste, e após a execução deles, foram encontrados 9 defeitos. A Tabela 5.7 apresenta o resumo das informações obtidas da *Sprint* 3.

Tabela 5.7 – Lista priorizada de casos de teste e defeitos encontrados na Sprint 3 (Estudo de Caso II).

#	Caso de teste	Importância	Defeitos
1	[QA] Agendamento	100%	MBA-1010, MBA-1030, MBA-1052
2	Tipo agendamento - CRUD	47.32%	MBA-755
3	[QA] Procedimento - CRUD	47.32%	MBA-675, MBA-676, MBA-678, MBA-836
4	[QA] Agendamento - Recorrência	39.44%	MBA-1003
5	[DEV] - Agendamento - Recorrências (MBA-899)	39.44%	
6	User Journey - Paciente em evidência	39.44%	
7	Medcall - receber ligação	39.44%	
8	Visualização atendimentos	19.72%	

Resultados

Os resultados calculados com a métrica APFD, a quantidade de casos de teste executados e a quantidade de defeitos encontrados de cada *sprint* e de cada abordagem de priorização são apresentados na Tabela 5.8.

Tabela 5.8 – Resultados do Estudo de Caso II.

	***Sprint* 1**			***Sprint* 2**			***Sprint* 3**		
Abordagem de Priorização	**APFD**	**Casos de teste**	**Defeitos**	**APFD**	**Casos de teste**	**Defeitos**	**APFD**	**Casos de teste**	**Defeitos**
Randômica	0,4393	9	11	0,3942	12	13	0,4236	8	9
Manual	0,55	6	5	0,6647	8	11	0,5	5	6
Proposta	0,5808	9	11	0,6506	12	13	0,7708	8	9
Ideal	0,7222	9	11	0,8173	12	13	0,8263	8	9

Análise dos Resultados

Na *Sprint* 1, a abordagem proposta apresentou valores melhores que as abordagens randômica e manual, como era esperado. Além disso, a abordagem manual deixou de encontrar 6 defeitos, porque deixou de executar 3 casos de teste. Isso ocorreu pelo fato da abordagem proposta adicionar à lista de casos de teste aqueles casos de teste que estão ligados a requisitos que tem dependência com o

requisito que está sendo testado. Em relação à priorização ideal, a abordagem proposta apresentou 80,4% de eficácia.

Na *Sprint* 2, o valor do APFD da abordagem proposta foi melhor que a abordagem randômica. Apesar de não tem sido melhor que a abordagem manual, por uma pequena diferença, a abordagem manual deixou de encontrar 2 defeitos por não ter executado 4 casos de teste. Comparando com a priorização ideal, a abordagem proposta apresentou 79,6% de eficácia.

E na *Sprint* 3, assim como na *Sprint* 1, a abordagem proposta apresentou um valor de APFD melhor do que as abordagens randômica e manual. E também, assim como nas *sprints* anteriores, a priorização manual deixou de encontrar defeitos que a abordagem proposta encontrou. Nessa *sprint* foram 3 defeitos e 3 casos de teste não executados. A porcentagem de eficácia da abordagem proposta em relação à priorização ideal foi de 93,2%.

Ameaças à Validade

Este estudo experimental também possui ameaças à validade, como por exemplo, a extração dos dados de entrada dos requisitos importados do Jira da Monitora. Como explicado anteriormente, essa extração foi feita de forma manual pelo autor desta dissertação e está suscetível a erros. Outro risco seria o mesmo do primeiro estudo de caso, em relação aos casos de teste elaborados, que podem não ter encontrado certos defeitos, o que causaria mudança nos valores obtidos de APFD.

5.4 Comparações de Resultados

Conforme explicado anteriormente, durante o levantamento bibliográfico deste trabalho, notou-se que muitos estudos da área fazem as validações de suas abordagens com programas disponibilizados em repositórios próprios para estudos experimentais controlados. Esses programas possuem versões com defeitos e também já possuem os casos de teste que revelam tais defeitos. O *Software-artifact*

Infrastructure Repository (2018) é um repositório que provê vários programas em linguagem C e Java para essa finalidade.

Como a abordagem proposta neste trabalho depende dos requisitos do sistema para priorizar os casos de teste, não foi possível realizar um estudo de caso com esses programas disponibilizados para experimentação. Porém, os estudos de caso realizados neste trabalho foram feitos com sistemas reais, o que eleva a dificuldade de sua condução e os riscos na obtenção dos resultados.

Ainda sim, decidiu-se fazer uma comparação dos resultados obtidos no segundo estudo de caso deste trabalho com outras abordagens de priorização de casos de teste encontradas na literatura. Foram selecionados alguns trabalhos que apresentam diferentes tipos de técnicas e grande relevância no mapeamento sistemático conduzido. Os trabalhos selecionados foram: Zhang et al. (2009), Carlson, Do e Denton (2011), Rajarathinam e Natarajan (2013), Indumathi e Selvamani (2015). Todos esses trabalhos utilizaram programas disponibilizados para experimentação.

A Tabela 5.9 apresenta os resultados desses quatro estudos e também os resultados da abordagem proposta por este trabalho. São considerados para comparação os valores de APFD mínimo, máximo e médio que cada abordagem obteve em seus respectivos estudos experimentais.

Tabela 5.9 – Comparação com outras abordagens de priorização.

Abordagem de Priorização	**APFD Mínimo**	**APFD Máximo**	**APFD Médio**
Proposta	0,5808	0,7708	0,6674
Zhang et al. (2009)	0,6071	0,8281	0,7492
Carlson, Do e Denton (2011)	0,4825	0,7392	0,6108
Rajarathinam e Natarajan (2013)	0,74	0,8375	0,7887
Indumathi e Selvamani (2015)	0,52	0,58	0,55

Como é possível notar, dois estudos apresentaram melhores resultados que a abordagem proposta, assim como outros que foram estudados durante este trabalho e que não foram selecionados para essa comparação, porém, os resultados obtidos pela abordagem proposta foram em estudos de caso com sistemas reais e também

foram bons resultados, apresentando um APFD médio maior que dois dos quatro estudos comparados.

5.5 Considerações Finais

Este capítulo apresentou os estudos experimentais conduzidos para avaliar a abordagem de priorização proposta e o *plug-in* desenvolvido para o Jira. O primeiro estudo de caso, com um sistema menor, apresentou resultados satisfatórios e também permitiu que alguns defeitos fossem corrigidos no *plug-in*. Já o segundo estudo de caso, feito em parceria com a Monitora, em um sistema bem maior apresentou bons resultados, que foram comparados com outras abordagens de priorização, e se aproximaram bastante da priorização ideal de cada *sprint*.

Também comparou-se os resultados obtidos nos estudos de caso com outras abordagens de priorização de casos de teste encontradas na literatura.

O próximo capítulo apresenta a conclusão deste trabalho.

Capítulo 6

CONCLUSÃO

Este capítulo apresenta as conclusões, contribuições e limitações deste trabalho de mestrado, assim como os possíveis trabalhos futuros.

6.1 Conclusões

Este trabalho de mestrado apresentou uma possível solução para a aplicação da estratégia de testes de regressão, que combina uma abordagem de priorização de casos de teste, baseada em rastreabilidade de requisitos e um *plug-in* para a ferramenta Jira. A abordagem de priorização proposta é uma evolução dos estudos feitos por Di Thommazo (2007, 2012, 2013a, 2013b, 2014a, 2014b, 2015), pois utiliza a Matriz de Rastreabilidade de Requisitos para priorizar os casos de teste.

Dois estudos de caso foram conduzidos para avaliar a eficácia da abordagem proposta. O primeiro estudo de caso foi realizado com um aplicativo de uma *start-up* que faz compartilhamento de mídias digitais. Os resultados obtidos foram satisfatórios e indicaram que a abordagem pode contribuir para a indústria. Esse estudo de caso também permitiu ajustes no *plug-in* desenvolvido. O segundo estudo de caso foi realizado em parceria com a empresa Monitora. O sistema utilizado foi o Medical Box, um sistema da área médica que vem sendo desenvolvido há mais de dois anos. Os resultados desse estudo de caso mostraram que a priorização feita pela abordagem proposta foi mais eficaz que a priorização feita de maneira manual pelos analistas de qualidade da Monitora, exceto em uma *sprint* por uma diferença muito pequena. Além disso, em todas as *sprints*, a priorização feita pela abordagem

proposta foi capaz de encontrar mais defeitos que a priorização manual dos analistas, porque considerou casos de teste que eles não consideraram.

6.2 Contribuições

As principais contribuições deste trabalho são:

- Definição de uma abordagem de priorização de casos de teste, baseada em rastreabilidade de requisitos;
- Desenvolvimento de um *plug-in* com suporte à abordagem de priorização proposta para uma ferramenta de ampla utilização no mercado – o Jira;
- Realização de estudos de caso com sistemas reais;
- Comparação dos resultados com outras abordagens de priorização encontradas na literatura.

6.3 Limitações

Algumas das limitações deste trabalho são:

- A abordagem de priorização utiliza apenas a Matriz de Rastreabilidade de Requisitos baseada em dados de entrada, que é uma das formas propostas por Di Thommazo (2014a) para gerá-la;
- Não foram exploradas outras formas de remover os casos de teste repetidos da lista de casos de teste de regressão dada pela abordagem;
- Os resultados obtidos nos estudos de caso não podem ser generalizados para qualquer sistema;
- O *plug-in* ainda precisa ser otimizado para trabalhar com múltiplos sistemas ao mesmo tempo.

6.4 Trabalhos Futuros

A seguir são apresentadas algumas possibilidades de melhoria deste trabalho de mestrado e também ideias de trabalhos futuros:

- Evoluir o *plug-in* desenvolvido para considerar as outras formas de geração da Matriz de Rastreabilidade de Requisitos: baseada em processamento de linguagem natural (RTM-NLP), em sistemas *fuzzy* (RTM-Fuzzy) e em redes neurais artificiais (RTM-N);
- Explorar outras formas de remover os casos de teste repetidos da lista de casos de teste de regressão dada pela abordagem;
- Propor uma nova abordagem de priorização de casos de teste, baseada em rastreabilidade de casos de teste.

REFERÊNCIAS

ABBOTT, R. J. **An integrated approach to software development**. John Wiley & Sons, 1986.

ABID, R., NADEEM, A. A novel approach to multiple criteria based test case prioritization. In: **13th International Conference on Emerging Technologies (ICET)**. IEEE, 2017.

ALSPAUGH, S. et al. Efficient time-aware prioritization with knapsack solvers. In: **Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007**. ACM, 2007. p. 13-18.

ANSARI, A. et al. Optimized Regression Test Using Test Case Prioritization. **Procedia Computer Science**, v. 79, p. 152-160, 2016.

ATLASSIAN. **Browse documentation**. ATLASSIAN Developer, 2017. Disponível em: <https://developer.atlassian.com/docs>. Acessado em: jan. 2018.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The Goal Question Metric Approach. In: **Encyclopedia of Software Engineering**. [S.l.: s.n.], 1994.

BELGAMO, A., FABBRI, S. C. P. F. GUCCRA: Técnica de Leitura para apoiar a Construção de Modelos de Caso de Uso e a Análise de Documentos de Requisitos. In: SBES – 19º SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. **Proceedings...** Uberlândia, MG, Brasil, 2005.

BLACK, R. **Managing the Testing Process**: practical tools and techniques for managing software and hardware testing. 3rd. ed. Indianapolis, IN: Wiley Publishing, 2009.

CARLSON, R.; DO, H.; DENTON, A. A clustering approach to improving test case prioritization: An industrial case study. In: **Software Maintenance (ICSM), 2011 27th IEEE International Conference on**. IEEE, 2011. p. 382-391.

CASTOR, A. et al. Towards Requirement Traceability in TROPOS. In: WORKSHOP EM ENGENHARIA DE REQUISITOS. **Proceedings...** 2004.

CATAL, C.; MISHRA, D. Test case prioritization: a systematic mapping study. **Software Quality Journal**, v. 21, n. 3, p. 445-478, 2013.

CLELAND-HUANG, J.; GOTEL, O.; ZISMAN, A. **Software and Systems Traceability**. [S.l.: s.n.], 2012.

CLELAND-HUANG, J. et al. Software Traceability: Trends and Future Directions. In: FOSE 2014 PROCEEDINGS OF THE ON FUTURE OF SOFTWARE ENGINEERING. **Proceedings…** 2014.

DI THOMMAZO, A. **Gerenciamento de Requisitos no Ambiente COCAR**. 2007. Dissertação (Mestrado em Ciências da Computação) – Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2007.

DI THOMMAZO, A. et al. Geração Automática da Matriz de Rastreabilidade de Requisitos com suporte de Visualização. In: SBES - SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. **Proceedings...** Natal, 2012.

DI THOMMAZO, A. et al. Detecting traceability links through neural networks. In: 25TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING. **Proceedings…** Boston, 2013a.

DI THOMMAZO, A. et al. An automatic approach to detect traceability links using fuzzy logic. In: 27TH BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING. **Proceedings…** Brasília, 2013b.

DI THOMMAZO, A. **Um Conjunto de Abordagens para a Geração da Matriz de Rastreabilidade de Requisitos com Suporte de Técnicas de Inteligência Computacional**. 2014. Tese (Doutorado em Ciências da Computação) – Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2014a.

DI THOMMAZO, A. et al. Using Artificial Intelligence Techniques to Enhance Traceability Links. In: 16TH INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEM. **Proceedings…** Lisboa, 2014b.

DI THOMMAZO, A. et al. Using the Dependence Level Among Requirements to Priorize the Regression Testing set and Characterize the Complexity of Requirements Change. In: 17TH INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEM. **Proceedings…** Barcelona, 2015.

DO, H. et al. An empirical study of the effect of time constraints on the cost-benefits of regression testing. In: **Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering**. ACM, 2008. p. 71-82.

DO, H. Recent Advances in Regression Testing Techniques. **Advances in Computers**, 2016.

ELBAUM, S.; MALISHEVSKY, A. G.; ROTHERMEL, G. **Prioritizing test cases for regression testing**. ACM, 2000.

ELBAUM, S.; MALISHEVSKY, A. G.; ROTHERMEL, G. Incorporating varying test costs and fault severities into test case prioritization. In: **Proceedings of the 23rd International Conference on Software Engineering**. IEEE Computer Society, 2001. p. 329-338.

ELBAUM, S.; MALISHEVSKY, A. G.; ROTHERMEL, G. Test case prioritization: A family of empirical studies. **IEEE transactions on software engineering**, v. 28, n. 2, p. 159-182, 2002.

FABBRI, S. C. P. F. et al. Managing literature reviews information through visualization. In: INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS, 14th. Jun 2012, Wroclaw. **Proceedings…** Lisboa: INSTICC, 2012. p. 36-45.

FAZLALIZADEH, Y. et al. Incorporating historical test case performance data and resource constraints into test case prioritization. In: **International Conference on Tests and Proofs**. Springer Berlin Heidelberg, 2009. p. 43-57.

GERACI, A. et al. **IEEE Standard Computer Dictionary**: compilation of IEEE standard computer glossaries. IEEE Press, 1991.

GUO, Y. et al. An ontology based improved software requirement traceability matrix. In: 2ND INTERNATIONAL SYMPOSIUM ON KNOWLEDGE ACQUISITION AND MODELING. **Proceedings…** 2009.

HASS, A. M. J. **Guide to advanced software testing**. 2nd. ed. Norwood, MA: Artech House, 2014.

HERNANDES, E. C. M. et al. Avaliação da ferramenta StArt utilizando o modelo TAM e o paradigma GQM. In: EXPERIMENTAL SOFTWARE ENGINEERING LATIN AMERICAN WORKSHOP, ESELAW, 10th. Nov., 2010, Goiânia. **Proceedings…** São Carlos: ICMC, 2010. pp. 10-22.

HETTIARACHCHI, C.; DO, H.; CHOI, B.. Risk-based test case prioritization using a fuzzy expert system. **Information and Software Technology**, v. 69, p. 1-15, 2016.

INDUMATHI, C. P.; SELVAMANI, K. Test Cases Prioritization Using Open Dependency Structure Algorithm. **Procedia Computer Science**, v. 48, p. 250-255, 2015.

JEFFREY, D.; GUPTA, N. Experiments with test case prioritization using relevant slices. **Journal of Systems and Software**, v. 81, n. 2, p. 196-221, 2008.

JIRA. **The #1 software development tool used by agile teams**. Jira Software, 2018. Disponível em: <https://www.atlassian.com/software/jira>. Acessado em: jan. 2018.

KAYES, I.; ISLAM, S.; CHAKARESKI, J. The network of faults: a complex network approach to prioritize test cases for regression testing. **Innovations in Systems and Software Engineering**, v. 11, n. 4, p. 261-275, 2015.

KHALILIAN, A.; AZGOMI, M. A.; FAZLALIZADEH, Y. An improved method for test case prioritization by incorporating historical test case data. **Science of Computer Programming**, v. 78, n. 1, p. 93-116, 2012.

KHIN, H. S. H.; YOUNGSIK, C.; JONG, S. P. Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting. In: **Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on**. IEEE, 2008. p. 527-532.

KIM, S.; BAIK, J. An effective fault aware test case prioritization by incorporating a fault localization technique. In: **Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement**. ACM, 2010. p. 5.

KIM, J.; PORTER, A. A history-based test prioritization technique for regression testing in resource constrained environments. In: **Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on**. IEEE, 2002. p. 119-129.

KLINDEE, P.; PROMPOON, N. Test cases prioritization for software regression testing using analytic hierarchy process. In: **Computer Science and Software Engineering (JCSSE), 2015 12th International Joint Conference on**. IEEE, 2015. p. 168-173.

KOREL, B.; TAHAT, L. H.; HARMAN, M. Test prioritization using system models. In: **21st IEEE International Conference on Software Maintenance (ICSM'05)**. IEEE, 2005. p. 559-568.

KOREL, B.; KOUTSOGIANNAKIS, G.; TAHAT, L. H. Application of system models in regression test suite prioritization. In: **Software Maintenance, 2008. ICSM 2008. IEEE International Conference on**. IEEE, 2008. p. 247-256.

KRISHNAMOORTHI, R.; SAHAAYA ARUL MARY, S. A. Requirement based system test case prioritization of new and regression test cases. **International Journal of Software Engineering and Knowledge Engineering**, v. 19, n. 03, p. 453-475, 2009.

KWON, J. et al. Test Case Prioritization Based on Information Retrieval Concepts. In: **2014 21st Asia-Pacific Software Engineering Conference**. IEEE, 2014. p. 19-26.

LEON, D.; PODGURSKI, A. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In: **Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on**. IEEE, 2003. p. 442-453.

LIN, C. et al. History-based test case prioritization with software version awareness. In: **Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on**. IEEE, 2013. p. 171-172.

MALDONADO, J. C. FABBRI, S. C. P. F. Teste de software. In: ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de Software**: teoria e prática. São Paulo: Prentice Hall, 2001.

MALDONADO, J. C. et al. **Introdução ao Teste de Software** - versão 2004-01. São Carlos: ICMC/USP, 2004.

MALZ, C.; GÖHNER, P. Agent-based test case prioritization. In: **2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops**. 2011.

MARTINS, M. D. C. **Geração de Pontos de Casos de Uso no Ambiente COCAR**. 2007. Dissertação (Mestrado em Ciências da Computação) – Departamento de Computação, Universidade Federal de São Carlos, São Carlos, 2007.

MEI, H. et al. A static approach to prioritizing junit test cases. **IEEE Transactions on Software Engineering**, v. 38, n. 6, p. 1258-1275, 2012.

MIRARAB, S.; TAHVILDARI, L. A prioritization approach for software test cases based on bayesian networks. In: **International Conference on Fundamental Approaches to Software Engineering**. Springer Berlin Heidelberg, 2007. p. 276-290.

MOHAPATRA, S. K.; PRASAD, S. Evolutionary Search Algorithms for Test Case Prioritization. In: **Machine Intelligence and Research Advancement (ICMIRA), 2013 International Conference on**. IEEE, 2013. p. 115-119.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. 3rd ed. Hoboken, NJ: John Wiley & Sons, 2011.

PARASHAR, P.; KALIA, A.; BHATIA, R. Pair-Wise Time-Aware Test Case Prioritization for Regression Testing. In: **International Conference on Information Systems, Technology and Management**. Springer Berlin Heidelberg, 2012. p. 176-186.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: 12TH INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING. **Proceedings…** Swinton, UK, UK: British Computer Society, 2008.

PRAKASH, N.; RANGASWAMY, T. R. Modular based multiple test case prioritization. In: **Computational Intelligence & Computing Research (ICCIC), 2012 IEEE International Conference on**. IEEE, 2012. p. 1-7.

PRESSMAN, R. S; MAXIM, B. R. **Software Engineering**: a practitioner's approach. 8th. ed. New York, NY: McGraw-Hill, 2015.

ROONGRUANGSUWAN, S.; DAENGDEJ, J. A test case prioritization method with practical weight factors. **J. Software Eng**, v. 4, p. 193-214, 2010.

ROTHERMEL, G. et al. Test case prioritization: An empirical study. In: **Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on**. IEEE, 1999. p. 179-188.

ROTHERMEL, G. et al. Prioritizing test cases for regression testing. **IEEE Transactions on software engineering**, v. 27, n. 10, p. 929-948, 2001.

SIDDIK, M. S.; SAKIB, K. RDCC: An effective test case prioritization framework using software requirements, design and source code collaboration. In: **Computer and Information Technology (ICCIT), 2014 17th International Conference on**. IEEE, 2014. p. 75-80.

SINGH, Y. et al. Systematic literature review on regression test prioritization techniques. **Informatica**, v. 36, n. 4, 2012.

SOMMERVILLE, I. **Software engineering**. 10th. ed. Boston, MA: Addison-Wesley, 2015.

SRIKANTH, H.; WILLIAMS, L.; OSBORNE, J. System test case prioritization of new and regression test cases. In: **2005 International Symposium on Empirical Software Engineering, 2005**. IEEE, 2005. p. 10 pp.

SRIKANTH, H.; BANERJEE, S. Improving test efficiency through system test prioritization. **Journal of Systems and Software**, v. 85, n. 5, p. 1176-1187, 2012.

SRIKANTH, H. et al. Towards the prioritization of system test cases. **Software Testing, Verification and Reliability**, v. 24, n. 4, p. 320-337, 2014.

SRIKANTH, H.; CASHMAN, M.; COHEN, M. B. Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study. **Journal of Systems and Software**, v. 119, p. 122-135, 2016.

STALLBAUM, H.; METZGER, A.; POHL, K. An automated technique for risk-based test case generation and prioritization. In: **Proceedings of the 3rd international workshop on Automation of software test**. ACM, 2008. p. 67-70.

STANDISH. **The Standish Group**. Disponível em: <http://standishgroup.com>. Acesso em: jan. 2018.

THOMAS, S. W. et al. Static test case prioritization using topic models. **Empirical Software Engineering**, v. 19, n. 1, p. 182-212, 2014.

TYAGI, M.; MALHOTRA, S. Test case prioritization using multi objective particle swarm optimizer. In: **Signal Propagation and Computer Technology (ICSPCT), 2014 International Conference on**. IEEE, 2014. p. 390-395.

VEDPAL; CHAUHAN, N.; KUMAR, H. A hierarchical test case prioritization technique for object oriented software. In: **Contemporary Computing and Informatics (IC3I), 2014 International Conference on**. IEEE, 2014. p. 249-254.

VESCAN, A., SERBAN, C., CHISALITA-CRETU, C., DIOSAN, L. Requirement dependencies-based formal approach for test case prioritization in regression testing. In: **13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)**. IEEE, 2017.

WANG, Z. et al. Cost-cognizant combinatorial test case prioritization. **International Journal of Software Engineering and Knowledge Engineering**, v. 21, n. 06, p. 829-854, 2011.

XIAO, L., MIAO, H., ZHUANG, W., CHEN, S. An empirical study on clustering approach combining fault prediction for test case prioritization. In: **16th International Conference on Computer and Information Science (ICIS)**. IEEE, 2017.

YOO, S.; HARMAN, M. Regression testing minimization, selection and prioritization: a survey. **Software Testing, Verification and Reliability**, v. 22, n. 2, p. 67-120, 2012.

ZAMBONI, A. B. et al. StArt Uma ferramenta computacional de apoio à Revisão Sistemática. In: CONGRESSO BRASILEIRO DE SOFTWARE – SESSÃO DE FERRAMENTAS, CbSoft, 1th. Set., 2010, Salvador. **Proceedings…** Salvador: UFBA, 2010. p. 10-12.

Apêndice A

PROTOCOLO DO MAPEAMENTO SISTEMÁTICO

O Apêndice A apresenta o protocolo elaborado para o mapeamento sistemático conduzido para este trabalho.

Objetivo: Fazer um levantamento das técnicas e ferramentas de apoio existentes para priorização de casos de teste.

Questão principal: Quais são as principais técnicas de priorização de casos de teste?

Questão secundária: Quais são as ferramentas de apoio para priorização de casos de teste?

Palavras-chave: priorização, prioritization, caso de teste, casos de teste, test case, test cases, software.

Critério de seleção das fontes: A fonte de busca deve exportar um arquivo .bib.

Lista das fontes: ACM, IEEE, Scopus e Web of Science.

Critérios de seleção dos estudos: Apresenta uma técnica de priorização de casos de teste (inclusão); Faz uma comparação entre técnicas de priorização de casos de teste (inclusão); Apresenta uma ferramenta que apóia a priorização de casos de teste (inclusão); Faz uma comparação entre ferramentas que apóiam a priorização de casos de teste (inclusão); Apresenta uma RS ou MS sobre priorização de casos de teste (inclusão); Não apresenta uma técnica ou ferramenta relacionada a priorização de casos de teste (exclusão); Anais de eventos (exclusão).

Formulário de extração de dados: Técnica utilizada; Grupo da técnica; Métrica utilizada; Utiliza alguma ferramenta?; Tipo de estudo; Resultado.