Resumo

Testes de software são conduzidos com o intuito de garantir a qualidade de um produto de software, verificar se o sistema projetado está de acordo com os requisitos do cliente e encontrar e reportar erros. Entretanto, o processo de teste é de alto custo, pois demanda muito tempo e esforço do projeto de software. Além disso, as atividades de teste são executadas habitualmente no final do processo de desenvolvimento, sendo frequentemente sacrificadas devido a atrasos no cronograma causados por outras atividades do projeto. Apesar do alto custo, a execução não apropriada das atividades de teste pode se tornar ainda mais cara, visto que capturar e corrigir um erro no ambiente do cliente pode custar até 100 vezes mais do que no ambiente de desenvolvimento.

A atividade de projeto de testes demanda uma grande parcela de tempo e esforço do processo de testes. O objetivo deste trabalho é diminuir o tempo necessário para a execução do projeto de testes, através da proposta de um algoritmo de geração automática de casos de teste. Reduzindo o tempo e o esforço do projeto de testes, consequentemente, reduziremos o custo do processo de testes.

O algoritmo proposto por este trabalho foi baseado em um processo de escrita manual de casos de teste. Para verificar o desempenho do algoritmo, foram realizados dois projetos de teste: o primeiro utilizando o algoritmo proposto e o segundo utilizando a abordagem manual. O projeto de testes utilizando a geração automática de casos de teste obteve uma redução de esforço de 58% comparado à abordagem manual.

Abstract

Software Tests are executed with the aim to guarantee the quality of a software product, to verify if the projected system is according to the customer requirements and to find and report bugs. Meanwhile, the test process is a high cost activity, because it demands a huge amount of time and effort of the software project. Also, the test activities are usually executed at the end of the software development, been often sacrificed because of schedule delays caused by other project activities. Despite the high cost, non-appropriate execution of the test activities can become even more expensive, since that capture an error at the customer environment can cost 100 times more than at the development environment.

The project tests activities demands a large amount of time and effort of the test process. The aim of this work is to lower the required time to execute the test project, through proposing an automatic test case generation algorithm. Reducing the time and effort of the test project, consequently, will reduce the cost of the software process.

The algorithm proposed by this work was based on a manual process of writing test cases. To verify the performance of the algorithm, two test projects were executed: the first one using the proposed algorithm and the second one using the manual approach. The test project using the automatic test case generation achieved a reduction of the effort about 58% compared to the manual approach.

Sumário

Índice de Figuras iv

Índice de Tabelas v

Tabela de Símbolos e Siglas vi

1 Introdução 8

1.1 Objetivos 9
1.2 Estrutura do Trabalho 9

2 Testes de Software 11

2.1 Abordagens e Técnicas de Testes 13
2.2 Ferramentas de Geração Automática de Casos de Teste 16

3 Java *Web* 18

4 Sistema de Cadastro de Casos de Uso 21

5 Geração Automática de Casos de Teste 24

5.1 Elaboração de Casos de Teste 25
5.2 Algoritmo de Geração Automática de Casos de Teste 29

6 Análise e Comparação dos Resultados 36

6.1 Metodologia 37
6.2 Resultados Obtidos 39
6.2.1 Esforço do projeto de testes 40
6.2.2 Número de casos de teste gerados 42
6.2.3 Número de casos de teste com possíveis registros duplicados 43
6.2.4 Qualidade dos testes 44
6.2.5 Cobertura dos testes 44
6.2.6 Impacto no processo de testes 45
6.2.7 Considerações Finais 45

7 Conclusões e Trabalhos Futuros 47

7.1 Contribuições 48
7.2 Dificuldades Encontradas 49
7.3 Trabalhos Futuros 50

Índice de Figuras

Figura 1. O papel de um *middleware* 18
Figura 2. Arquitetura do Sistema de Cadastro de Casos de Uso 23
Figura 3. Um Caso de Uso e seus Fluxos 27
Figura 4. Exemplo de Caso de Uso Complexo 28
Figura 5. Possíveis Casos de Teste 30

Índice de Tabelas

Tabela 1. Caso de uso Fazer Pedido 26
Tabela 2. Exemplo de Caso de Teste 28
Tabela 3. Fluxo alternativo Fornece CEP 31
Tabela 4. Fluxo alternativo Adicionar Item 31
Tabela 5. Fluxo de exceção Informação Incorreta 32
Tabela 6. Fluxo de Fornecer CEP 32
Tabela 7. Fluxo de Adicionar Item 32
Tabela 8. Fluxo de Informação Incorreta 33
Tabela 9. Caso de Teste Fazer Pedido com Sucesso 34
Tabela 10. Caso de Teste Fornecer CEP 34
Tabela 11. Caso de Teste Adicionar Item 35
Tabela 12. Caso de Teste Informação Incorreta 35
Tabela 13. Abordagem Manual 40
Tabela 14. Abordagem Automática 40
Tabela 15. Número de Testes Gerados 42
Tabela 16. Número de Testes com possíveis registros duplicados 43

Tabela de Símbolos e Siglas

AETG – *Automatic Efficient Test Generator*

DGL – *Data Generation Language*

UniTesk – *Unified Test and Specification Toolkit*

ASML – *Abstract State Machine Language*

XML – *eXtensible Markup Language*

JSP – *Java Server Pages*

API – *Application Programming Interface*

JSE – *Java Standard Edition*

JEE – *Java Enterprise Edition*

HTML – *HyperText Markup Language*

HTTP – *Hypertext Transfer Protocol*

XP – *Extreme Programming*

IDE – *Integrated Development Environment*

MBT – *Model-based Testing*

TI – Tecnologia da Informação

Agradecimentos

Primeiramente eu gostaria de agradecer a Deus, que me guiou e iluminou ao longo deste trabalho e de todo o curso de graduação. Gostaria de agradecer a minha família, aos meus pais, Adeildo José de Freitas e Maria Eunice Dornelas de Freitas e meu irmão, Valdner Dornelas de Freitas, que me apoiaram e confortaram durante todo curso e toda minha vida. Sem eles, com certeza eu não estaria aqui.

Agradeço especialmente a minha namorada, Carolina Valentine Magalhães Pollari, minha melhor amiga, companheira e cúmplice. Graças ao seu amor, carinho e compreensão eu encontrei forças pra superar as dificuldades. Sem o apoio dela e seus "puxões de orelha" eu não teria conseguido terminar este trabalho.

Gostaria de agradecer também aos meus amigos e colegas de faculdade, com os quais convivi e aprendi muito ao longo do curso. São tantos os amigos que é inviável dizer todos os nomes, mas agradeço a cada uma da pessoas que conheci durante o curso. Todos me ajudaram diretamente ou indiretamente a chegar aonde eu cheguei.

Agradeço também ao professor Márcio Lopes Cornélio, por sua orientação, compreensão, dedicação e seus grandes conselhos. Devo muito da minha graduação e do meu trabalho de conclusão a ele.

Gostaria de agradecer ao mestrando Lúcio Ribeiro pela grande ajuda que ele me forneceu para compreender e conduzir este trabalho. Agradeço também a mestranda Maíra Paschoalino por ajudar na proposta do tema deste trabalho.

Agradeço aos meus colegas de trabalho da TCI. Sem a compreensão e o apoio deles a finalização deste trabalho não seria possível. Gostaria de agradecer também aos meus colegas do Projeto CIn/Motorola que contribuíram bastante para o meu amadurecimento e aprendizagem tanto pessoal quanto profissional.

E por fim, mas não menos importante, agradeço a todos os professores do DSC. O empenho, a seriedade e o exemplo de ética e profissionalismo deles contribuíram bastante para a minha formação pessoal e profissional.

Capítulo 1

Introdução

O processo de testes exerce um papel fundamental na garantia da qualidade dos projetos de software atualmente. Um projeto de um software se inicia com o levantamento dos requisitos do cliente. Com o objetivo de garantir que os requisitos do cliente sejam atingidos satisfatoriamente, a atividade de testes elabora um conjunto de casos de teste para verificar e validar o sistema desenvolvido. Quando projetados e executados da maneira adequada, os testes são capazes de assegurar que o sistema foi desenvolvido com qualidade, de acordo com os critérios de aceitação previamente estabelecidos, e que atende as expectativas do cliente.

Testes procuram encontrar erros e comportamentos que não estão de acordo com os requisitos para que estes sejam corrigidos ainda durante a fase de desenvolvimento. Entretanto, a atividade de testes requer muito esforço, consome muitos recursos do projeto de software e é uma atividade de alto custo. Estudos indicam que o processo de testes pode consumir quase 50% dos recursos de um projeto [2]. Entretanto, o prejuízo de não executar esta atividade ou de não executá-la apropriadamente é muito maior. O custo de capturar um erro após a homologação do sistema com o cliente e a implantação deste sistema no ambiente de produção pode ser até 100 vezes maior do que a captura deste erro durante a fase de desenvolvimento [7].

A atividade de testes procura reproduzir o ambiente do usuário e simular o que pode ocorrer durante o uso normal do sistema para verificar o seu comportamento. O processo de testes segue um conjunto de atividades bem definidas, procurando encontrar erros no sistema o mais rápido possível reduzindo os custos de manutenção do software após a entrega para o cliente.

Procurar maneiras de reduzir o custo da atividade de testes é muito importante para garantir que essa atividade seja cada vez mais adotada e para que não ocorram atrasos nas entregas do projeto. Este trabalho busca minimizar o custo das atividades de teste através da

proposta de um algoritmo de geração automática de casos de teste, baseada nos requisitos funcionais do sistema, mais especificamente casos de uso, para verificar se o sistema se comporta adequadamente e encontrar erros. Reduzindo o esforço da atividade de projeto de testes, é possível obter uma redução total no custo do processo de testes, visto que o projeto de testes é uma atividade que demanda muito esforço da equipe de testes.

1.1 Objetivos

O objetivo geral deste trabalho é contribuir para a redução do custo do processo de testes. Durante o estudo deste trabalho, percebeu-se que a atividade de projeto de testes é uma atividade que corresponde a uma grande parcela do custo de testes e que o custo desta atividade pode ser reduzido. Por se tratar de uma atividade muitas vezes repetitiva, isso indica uma oportunidade para automatização desta atividade.

O objetivo principal deste projeto é propor um algoritmo de geração automática de casos de teste baseado em casos de uso. Com a utilização deste algoritmo supõe-se que a atividade de projeto de testes terá uma significativa redução. Para verificar se o algoritmo realmente minimiza o custo dos projetos de testes, um experimento será conduzido através da execução de um mesmo projeto de testes utilizando a abordagem manual de elaboração de casos de testes e a abordagem de geração automática de casos de teste proposta. Como parâmetros de comparação entre as abordagens, alguns métricas serão levantadas e discutidas com o intuito de avaliar o desempenho da abordagem automática perante a abordagem manual.

1.2 Estrutura do Trabalho

Além deste capítulo introdutório, este trabalho é composto por mais seis capítulos. No Capítulo 2 apresentaremos o que é a atividade de Testes de Software, bem como uma história sucinta do desenvolvimento da atividade de testes. Também discutiremos as principais abordagens e técnicas de testes utilizadas. Por fim, apresentaremos algumas ferramentas de geração automática de casos de teste e descreveremos brevemente como elas funcionam.

No Capítulo 3 apresentaremos a tecnologia Java *Web*. Essa tecnologia foi utilizada para implementação do algoritmo de geração automática de casos de teste proposto por estes trabalho.

O Capítulo 4 trata do Sistema de Cadastro de Casos de Uso, o sistema para o qual o módulo de geração automática de casos de teste foi desenvolvido. Descreveremos a arquitetura

do sistema, seus objetivos principais, falaremos da equipe que o desenvolveu e das dificuldades que enfrentaram durante o seu desenvolvimento.

No Capítulo 5 abordaremos a Geração Automática de Casos de Teste. Iniciaremos discutindo como funciona a elaboração manual dos casos de testes na qual este trabalho se baseou e por fim descreveremos o funcionamento do algoritmo de geração automática de casos de teste proposto.

No Capítulo 6 discutiremos como foi conduzido o estudo comparativo entre as abordagens manual e automática. Descreveremos as métricas utilizadas na comparação e o porquê de terem sido escolhidas. Também apresentaremos os resultados obtidos durante os experimentos e, por fim, discutiremos o desempenho da abordagem de geração automática proposta comparada com a elaboração manual.

O Capítulo 7 conclui este trabalho apresentando as nossas principais contribuições. Neste capítulo também relatamos as principais dificuldades enfrentadas e finalizamos propondo diversas oportunidades de trabalhos futuros a partir deste projeto.

Capítulo 2

Testes de Software

Testes são fundamentais para garantir a qualidade dos produtos de software [2]. O objetivo de executar testes é capturar falhas ou erros em um software utilizando um conjunto de atividades e uma seqüência de passos bem planejados [5]. A atividade de testes também é definida como uma área da Engenharia de Software que procura melhorar a produtividade e procurar evidências de confiabilidade e qualidade de software como um complemento a outras atividades de qualidade inseridas no processo de desenvolvimento de software [6].

O processo de desenvolvimento de software pode introduzir diversos erros e a execução de testes procura capturar a maior quantidade possível de defeitos para verificar se os requisitos do sistema foram corretamente implementados. Além de buscar assegurar a qualidade e corretude do software produzido, testes podem reduzir os custos de manutenção corretiva e aumentar a satisfação do cliente [1]. No entanto, é importante salientar que o processo de testes não garante a total ausência de erros.

No início da história do desenvolvimento de software, os sistemas eram mais simples e os testes estavam focados no hardware. Os problemas no software eram considerados menos relevantes. As atividades de teste eram completamente *ad hoc* e realizadas pelos desenvolvedores com base nos seus conhecimentos sobre o sistema. Posteriormente, o conceito de *debugging* começou a ser considerado como uma atividade distinta da atividade de testes. *Debugging* consiste em procurar e corrigir erros em programas e está mais associada a uma atividade de desenvolvimento. Já as atividades de teste passaram a verificar se o programa funcionava de acordo com o que foi especificado pelo cliente. As aplicações se tornavam cada vez mais complexas, e as atividades de testes ganhavam maior relevância [8].

No fim da década de 1970, as atividades de teste foram definidas com um processo de execução de um programa com o intuito de encontrar falhas. Tendo como o objetivo demonstrar a ausência de falhas, o testador pode ser levado a escrever e executar testes com baixa probabilidade de falhar. Mudando o foco dos testes de provar a ausência de falhas para encontrar falhas, há uma tendência de que os testes sejam elaborados e executados de forma mais eficiente [1]. Na década de 1980, foram definidas metodologias, dividindo o processo de testes em atividades de análise, revisão e execução, com o intuito de avaliar o software durante todo o seu ciclo de vida.

Atualmente a atividade de teste desempenha um papel fundamental para o sucesso de um projeto de desenvolvimento de software. Testes passaram a ser um processo paralelo ao de desenvolvimento, com suas próprias atividades de planejamento, análise de requisitos de testes, projeto, implementação de *scripts*, execução e manutenção. Entretanto, estudos indicam que as atividades de um processo de testes exigem de 45% a 50% dos recursos de um projeto [2]. Maiores informações a respeito das atividades de um processo de teste serão descritas na próxima seção.

Apesar do elevado custo de testar softwares, o prejuízo de não executar essa atividade pode ser muito maior. Estima-se que o custo para se detectar um erro após a entrega do produto, ou seja, após a implantação no ambiente de produção, é cerca de 100 vezes maior do que a realização de testes para a captura do mesmo erro durante a fase de desenvolvimento [7].

A atividade de testes costuma ser executada, em geral durante toda a fase de desenvolvimento pelos próprios desenvolvedores. No entanto, a maior carga de testes tende a ser executada nas etapas finais do processo de desenvolvimento. A atividade de testes demanda muito tempo e esforço, como visto anteriormente, entretanto o tempo para a execução da atividade de testes é frequentemente reduzido devido a atrasos em atividades anteriores do projeto, resultando em testes insuficientes de produtos antes de seu lançamento [4].

Por ser uma atividade que demanda muito esforço e recursos do projeto, a atividade de testes não pode ser executada de maneira irresponsável. Para que testes sejam executados da melhor maneira possível, otimizando os recursos e descobrindo o maior número de defeitos se faz necessário a aplicação sistemática de estratégias. Uma estratégia fornece um roteiro com os passos necessários para os testes serem conduzidos, planejamento de quando os testes serão projetados e executados e a quantidade de esforço, tempo e recursos requeridos [5].

2.1 Abordagens e Técnicas de Testes

É importante salientar que existem duas abordagens fundamentais de teste: funcional e estrutural. A abordagem funcional, também conhecida como testes caixa-preta (*black-box*), tem o principal foco na verificação e validação dos requisitos funcionais do sistema, sem se preocupar com a forma como o sistema realiza a funcionalidade. Esses tipos de testes são de mais alto-nível, avaliando se as saídas produzidas estão de acordo com as entradas fornecidas. Os testes caixa-preta costumam ser executados principalmente durante o final do processo de testes. A abordagem estrutural, também conhecida como caixa-branca (*white-box*), considera principalmente aspectos e comportamentos internos e caminhos lógicos do sistema, tendo como principal foco a arquitetura e a lógica do código. Testes caixa-branca são executados normalmente durante a etapa de desenvolvimento, pelos próprios desenvolvedores do código, pois é necessário o conhecimento da estrutura interna e da lógica da aplicação [1].

Os testes podem ser classificados como fazendo parte de três grandes etapas durante o desenvolvimento: testes unitários, testes de integração e testes de sistema. Os testes unitários avaliam o funcionamento de partes isoladas do software, como componentes ou sub-rotinas. Testes de integração validam a interação entre diferentes componentes e são executados sempre que há inserção de novos componentes. Os testes de sistema verificam o comportamento do sistema como um todo, tanto requisitos funcionais como não-funcionais [9].

Existem vários tipos de testes de sistema, entre eles estão: testes de recuperação, testes de segurança, testes de estresse e testes de desempenho. Testes de recuperação procuram levar o software a falhar de diversas formas e verificar se o seu comportamento diante dessas falhas é adequado. Testes de segurança exercitam os mecanismos de proteção do sistema diante de uma invasão externa. Testes de estresse verificam o comportamento do sistema quando a demanda de recursos, tanto em quantidade quanto em freqüência e volume são anormais. Os testes de desempenho procuram avaliar se o desempenho do sistema durante a execução é satisfatório e esta de acordo com o que foi projetado [5].

Em relação à qualidade dos testes executados, alguns critérios importantes devem ser considerados. Os testes devem ter alta probabilidade de encontrar erros, ou seja, o sistema deve ser bem compreendido para que os testes projetados tenham capacidade de exercitar situações em que o software pode falhar. Testes não devem ser redundantes, pois os recursos de um projeto de software são limitados e não justifica o projeto e a execução de testes que possam capturar o mesmo defeito. Quando há um conjunto de testes que possuem um objetivo semelhante, deve-se

procurar escolher um subconjunto de testes que tenha a maior probabilidade de revelar erros. Preferencialmente os testes devem ser executados separadamente, em pequenas granuralidades, procurando verificar comportamentos específicos. É importante evitar a tentação de se combinar vários testes em um mesmo caso de teste [2].

Um caso de teste é uma seqüência de passos que deve ser executada por um testador com o intuito de verificar se o software desenvolvido está de acordo com a especificação do sistema. Para cada passo do caso de teste há um resultado esperado. O resultado esperado descreve a resposta do sistema. Para cada caso de teste é necessário satisfazer uma série de pré-condições ou pré-requisitos. Esses pré-requisitos para execução também são descritos no caso de teste antes dos passos que serão executados.

Após a execução dos casos de teste, o resultado desta execução deve ser reportado. Caso o sistema se comporte de acordo com os resultados esperados dos passos do caso de teste, é dito, comumente, que o teste “Passou”. Se uma ou mais respostas do sistema não estiverem de acordo com os resultados esperados, diz-se que o teste “Falhou” e um registro de defeito é aberto para informar a equipe de desenvolvimento. No caso de algum caso de teste possuir como pré-requisito a execução de um caso de teste que “Falhou”, esse caso de teste não é executado. O caso de teste é “Bloqueado” e o registro de defeito aberto para o caso de teste do pré-requisito é associado ao teste “Bloqueado”.

Neste trabalho, a abordagem de testes que será estudada com mais profundidade é a abordagem caixa-preta. Como exposto anteriormente, testes caixa-preta procuram avaliar o comportamento do sistema em relação as suas funcionalidades de alto-nível. A classe de erros encontrada por esse tipo de teste é diferente da encontrada na abordagem caixa-branca. Com testes caixa-preta é possível encontrar erros relativos à falta ou funcionalidades implementadas incorretamente, erros de interface, erros de acesso a meios de armazenamento de dados, erros de comportamento e problemas de desempenho [5].

Entre as diversas técnicas de testes caixa-preta podemos citar: Testes de Regressão, Testes de Valor-limite, Particionamento de Equivalência, Testes de Exceção, Testes de Matriz Ortogonal e Testes Exploratórios.

Testes de Regressão são testes executados para avaliar se mudanças efetuadas devido ao desenvolvimento de novas funcionalidades de software não resultaram em inserção de erros em funcionalidades testadas anteriormente. O ideal seria executar todos os testes (Regressão total) após cada fase de desenvolvimento, para verificar se no sistema não surgiu nenhum efeito-

colateral inesperado. Entretanto devido ao tempo e custo, os testes de regressão escolhidos costumam ser os relacionados diretamente com as modificações adicionadas [9].

Testes de Valor-limite também são conhecidos como testes de fronteira. Esse tipo de teste procura verificar os valores limites das entradas e saídas, como de fronteiras da interface e se baseia no argumento de que defeitos têm maior probabilidade de serem encontrados nas fronteiras dos objetos e dos dados [11].

Particionamento de Equivalência é um método que divide o domínio de entrada em classes. A partir dessas classes são definidos os casos de teste. O objetivo seria projetar um caso de teste que fosse capaz de revelar toda uma classe de erros. Um benefício direto desse método é a redução na quantidade de casos de teste, obtendo uma redução nos custos [5].

Testes de Exceção procuram verificar as mensagens de erro e as condições que as ativam. Essa técnica pode ser aplicada inclusive em testes unitários, forçando o software a executar todos os caminhos onde as exceções serão lançadas [10].

Testes de Matriz Ortogonal são aplicados em situações em que o domínio da entrada é pequeno. Essa técnica ajuda a identificar erros relacionados a uma categoria de erros [5].

Testes Exploratórios podem ser visto como uma aliança entre as atividades de projeto e de execução de testes. Essa técnica não se baseia em um conjunto de passos definidos anteriormente, ela se baseia em que áreas do produto devem ser testadas, ou qual estratégia será aplicada. Entre as vantagens dessa técnica está a possibilidade de se verificar comportamentos que não são testados por casos de teste previamente projetados e auxiliar em situações onde o sistema não é completamente conhecido e encontrar os defeitos mais importantes mais rapidamente [12].

Para um conjunto de testes ser considerado completo, é preciso haver pelo menos um teste para cada fluxo de execução de cada funcionalidade ou caso de uso do sistema. A quantidade mínima de testes seria equivalente à quantidade de caminhos possíveis de execução do software.

Como exposto anteriormente, o processo de testes demanda uma grande quantidade de esforço. As atividades que geralmente compõem um processo de testes podem ser divididas em: planejamento, projeto, implementação, execução e avaliação. O planejamento de testes consiste em priorizar os requisitos a serem testados, elaborar estratégias de testes, produzir um cronograma, estimar esforço e quantidade de recursos necessários. O projeto é caracterizado pela elaboração de casos de testes, procedimentos para execução dos casos de teste e preparação do ambiente de testes. A etapa de implementação consiste em implementar *scripts* e componentes de teste. Na execução, os testes elaborados na etapa de projeto são executados manualmente ou automaticamente com o auxílio dos *scripts* criados na etapa de implementação. E, por fim, é feita

a avaliação dos testes onde é verificada a cobertura dos testes, a tendência dos defeitos e os critérios de sucesso [2].

2.2 Ferramentas de Geração Automática de Casos de Teste

Uma ferramenta geradora de casos de teste pode ser definida como a implementação de um processo automatizado que recebe como parâmetros de entrada um modelo do comportamento do software que será testado e um conjunto de diretivas para geração de casos de testes que auxiliam a ferramenta durante o processo de geração.

Existem várias ferramentas, as quais serão citadas posteriormente, nas quais as diretivas para geração de casos de teste não são fornecidas de forma explícita. Muitas vezes essas diretivas já estão encapsuladas na estrutura da ferramenta. A ferramenta desenvolvida neste trabalho se encaixa nessa categoria. Nessa categoria, o projetista não fornece as diretivas como entrada da ferramenta, mas a lógica e a estratégia de geração de casos de teste é implementada internamente na ferramenta de forma transparente para o projetista.

Assim como no processo manual de geração de casos de teste, no processo automático de geração, utilizando uma ferramenta de suporte, a saída da ferramenta é um conjunto de casos de teste, que incluem uma seqüência de passos que representam as ações do usuário sobre o sistema e as respostas esperadas dessas ações, de acordo com o documento de requisitos do software que será testado.

É importante salientar que existe uma diferença fundamental entre ferramentas de geração de casos de teste baseada em modelos e ferramentas de geração de entradas baseada em modelos. Esta última classe de ferramentas não é capaz de gerar os resultados esperados do sistema testado. Os modelos utilizados por essas ferramentas descrevem apenas as seqüências de entradas aceitas pelo sistema, mas não modela as saídas. Dois exemplos de ferramentas de geração de entradas baseada em modelos são: AETG (*Automatic Efficient Test Generator*) da Telcordia Technologies [25] e DGL (*Data Generation Language*) de Peter Maurer [26].

Também é importante diferenciar ferramentas de geração automática de casos de teste de *frameworks* de automação de testes. Um *framework* de automação de testes aceita como entrada testes gerados manualmente, gerados automaticamente ou seqüências de testes pré-gravadas e executa a seqüência de passos do testes de maneira automatizada sem a necessidade de intervenção ou supervisão humana. Exemplos conhecidos de *frameroks* de automação são:

WinRunner da Mercury [27], Rational Robot da IBM Rational [28], Tau Tester da Telelogic [29] e Test Complete da AutomatedQA [30].

Entre as ferramentas comerciais de geração automática de casos de teste podemos citar: TVEC [31], Conformiq Test Generator [32], Reactis [33], e Unitesk [34]. A ferramenta TVEC recebe como entrada modelos de requisitos e comportamentos especificados na linguagem proprietária T-VEC *Linear Form*. Esta ferramenta é usada principalmente na indústria aeroespacial e não fornece nenhum suporte para UML ou qualquer linguagem de modelagem amplamente utilizada. Já a ferramenta da Conformiq, aceita como entrada modelos UML. A ferramenta Reactis da *Reactive Systems* aceita modelos em Simulink [35] e na linguagem de modelagem StateFlow [36]. A ferramenta UniTesk (*Unified Test and Specification Toolkit*) se encontra no limite entre ferramenta comercial e ferramenta acadêmica. Os modelos são especificados em linguagens de especificação projetadas para programas Java e C++. Essas especificações são documentadas como comentários nos códigos das classes e métodos que serão testados.

Existe uma ferramenta (ainda sem um nome definido) de geração de casos de teste que funciona com ASML (*Abstract State Machine Language*) [37]. ASML é uma linguagem de especificação formal executável baseada na teoria de máquinas de estado abstratas. Ela utiliza XML (*eXtensible Markup Language*) e o Microsoft Word para descrever as especificações. O algoritmo de geração desta ferramenta é derivado de uma máquina de estados finitos. Essa ferramenta não está disponível comercialmente. Maiores informações sobre essa ferramenta podem ser encontradas em [38].

Capítulo 3

Java *Web*

A denominação Java *Web* se refere ao modelo de programação utilizando a tecnologia Java, voltado para aplicações *Web*. *Servlets* e *Java Server Pages* são os programas Java que executam em um servidor *Web*, e respondem a requisições feitas por clientes através de navegadores. *Servlets* e *Java Server Pages* (JSP) têm sido utilizados amplamente em aplicações *Web* das mais diferentes naturezas e funções, desde lojas *on-line* a qualquer tipo de aplicação interativa [13]. A Figura 1 mostra alguns dos papéis que podem ser assumidos por um *middleware*.

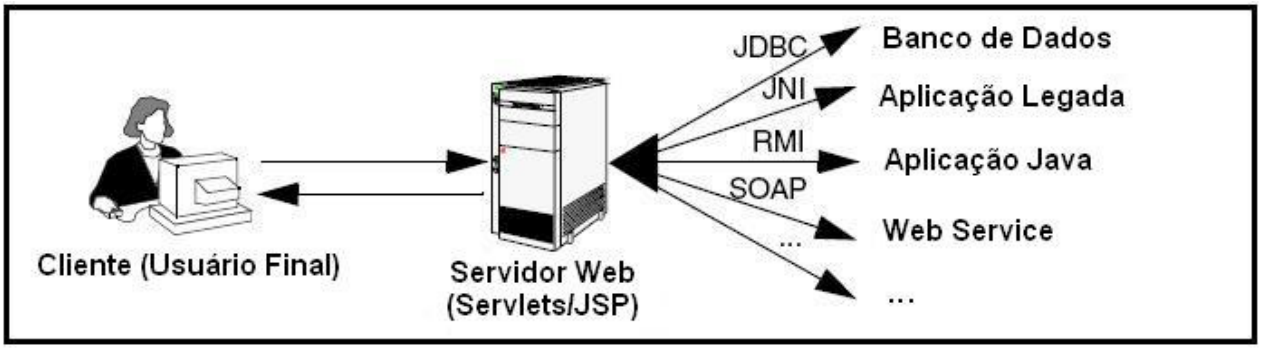

Figura 1. O papel de um *middleware*

Servlets, como dito anteriormente, são programas Java comuns, ou seja, a sintaxe de um *servlet* é a mesma de um programa Java convencional. Entretanto, novas APIs (*Application Programming Interface*) foram adicionadas na plataforma JEE (*Java Enterprise Edition*) para dar suporte a esse tipo de programa. Essas novas APIs não fazem parte da plataforma Java padrão (JSE – *Java Standard Edition*). Elas constituem uma especificação separada que faz parte apenas da plataforma JEE.

Servlets podem ser vistos como programas Java com código HTML (*HyperText Markup Language*) embutido, ou seja, o código é essencialmente Java, mas possui trechos de código

escritos em HTML. Documentos JSP podem ser vistos como páginas HTML com código Java embutido. Em essência, JSP é uma forma de se escrever um *servlet*. As páginas JSP são traduzidas em *servlets*, e estes são compilados e executados quando requisitados. O principal objetivo de JSP é simplificar a edição e manutenção de páginas HTML. Já os *servlets* são úteis para desenvolver a lógica do negócio das aplicações. As duas tecnologias podem ser utilizadas de maneira integrada, ou separadamente. Em aplicações cujo foco é apresentação, JSP se configura como a melhor opção. Em aplicações orientadas a tarefas, a melhor opção é utilizar *servlets*.

Servlets são capazes de ler dados enviados por um formulário HTML em uma página *Web*, gerar resultados e enviar dados para o cliente. Também podem tratar requisições GET (tipo mais comum de requisições para páginas *Web*) e requisições POST (requisições geradas quando um formulário HTML que especifica METHOD= “POST” é submetido).

Em termos de programação, todos os *servlets* estendem (herdam comportamento) de uma classe padrão conhecida como HttpServlet, que oferece suporte e infra-estrutura para responder a requisições HTTP (*Hypertext Transfer Protocol*). Na prática, os *servlets* sobrescrevem o método *doGet* ou *doPost*, dependendo de que dado está sendo enviado através de GET ou POST, e especificam diferentes métodos para responder a diferentes tipos de comandos HTTP. Se for necessário executar a mesma tarefa para os dois métodos, *doGet* e *doPost*, basta fazer com que o método *doGet* chame o *doPost*, ou vice-versa.

Os métodos *doGet* e *doPost* recebem dois argumentos: HttpServletRequest e HttpServletResponse. O argumento HttpServletRequest, permite o recebimento de todos os dados de entrada, nessa classe existem métodos que permitem o tratamento de todas as informações como formulários, cabeçalhos e o *hostname* do cliente. O argumento HttpServletResponse permite a especificação das informações de saída como códigos de *status* HTTP e cabeçalhos de resposta.

Quando um *servlet* é criado, o seu método *init* é invocado. O método *init* é onde está contido o código de configuração. Depois da invocação do método *init*, para cada requisição do usuário um *thread* é criado. Esse *thread* chama o método *service* da instância criada anteriormente. O método *service* chama os métodos *doGet*, *doPost* ou outro método dependendo do tipo de requisição HTTP recebida.

A tecnologia JSP permite aliar páginas de conteúdo estático com conteúdo gerado dinamicamente. Primeiramente é necessário escrever uma página HTML, em seguida o código de conteúdo dinâmico deve ser encapsulado com *tags* especiais, geralmente começando com ‘<%’ e terminando com ‘%>’.

S*ervlets* e JSP são tecnologias que apesar das diferenças, são em essência equivalentes. Entretanto, como dito anteriormente, elas não são úteis em todos os tipos de situações. *Servlets* são ótimos para programação e processamento de dados, mas apresentam diversas fraquezas quando se trata de apresentação destes dados. Utilizando *servlets*, é difícil manter e escrever código HTML, pois o código não se parece com HTML, então é mais difícil de ser visualizado como tal, tornando-o inacessível para programadores *web* que desconhecem Java. Além disso, programando *servlets* não é possível utilizar ferramentas de desenvolvimento *web*.

Com JSP é mais fácil escrever e manter código HTML, pois seu código estático é essencialmente HTML, sem a utilização de características da sintaxe de Java. Também é possível dividir a equipe de desenvolvimento. Os programadores Java podem trabalhar no código dinâmico, enquanto os desenvolvedores *web* se concentram na camada de apresentação.

Capítulo 4

Sistema de Cadastro de Casos de Uso

O Sistema de Cadastro de Casos de Uso foi desenvolvido durante a disciplina de Engenharia de Software de uma turma de mestrado do Departamento de Sistemas e Computação (DSC). O objetivo da disciplina era avaliar o uso de metodologias ágeis [15] em um projeto de desenvolvimento de Software. O objetivo do projeto era desenvolver uma ferramenta *Web* que fornecesse suporte a atividade de elaboração de casos de teste, mais especificamente a geração automática de casos de teste [14].

O sistema foi desenvolvido tendo por ponto de partida duas visões distintas: a do analista de sistemas e do analista de testes. Na visão do analista de sistema, foi percebida a necessidade do desenvolvimento de uma ferramenta de manutenção de casos de uso. Um dos benefícios da ferramenta é a padronização da escrita dos documentos de casos de uso e a maior disponibilidade dessas informações para os membros da equipe envolvidos. Na visão do analista de testes, o objetivo era fornecer uma ferramenta que reduzisse o tempo e o custo da elaboração de casos de teste.

O Sistema de Cadastro de Casos de Uso foi desenvolvido de maneira iterativa e incremental. A equipe de desenvolvimento foi formada por sete estudantes de mestrado, onde estes foram divididos em pares, ficando um membro como curinga e realizando diversas atividades ao longo do projeto. Com o objetivo de facilitar o desenvolvimento e dividir o projeto em iterações, o sistema foi dividido em diversos módulos:

- Módulo de cadastro de projeto: responsável por manter o cadastro dos projetos;
- Módulo de cadastro de requisito: responsável por manter os requisitos dos projetos;

- Módulo de cadastro de ator: responsável pelo cadastro de atores de cada caso de uso do sistema;
- Módulo de cadastro de campos: responsável pelo cadastro de campos de entrada e saída de cada caso de uso;
- Módulo de cadastro de fluxo geral: consiste no cadastro de fluxos gerais que podem ser aplicados a vários casos de uso;
- Módulo de cadastro de caso de uso: principal funcionalidade do sistema, onde todos os outros módulos anteriores são relacionados, sendo responsável por manter o cadastro dos casos de uso de cada projeto;
- Módulo de geração de caso de teste: responsável pela geração automática de casos de teste a partir da seleção de casos de uso e manipulação dos mesmos.

Para o desenvolvimento do Sistema de Cadastro de Casos de Uso, foi utilizada a metodologia ágil *Extreme Programming* (XP) [16]. Um protótipo foi desenvolvido no início do projeto e foi utilizado como guia ao longo de todo desenvolvimento. É importante ressaltar que durante o desenvolvimento a equipe enfrentou diversos problemas, como dificuldade para configuração do ambiente de desenvolvimento, falta de conhecimento da tecnologia utilizada e adaptação à metodologia de desenvolvimento.

A Figura 2 exibe a arquitetura do sistema de cadastro de casos de uso em alto nível. O sistema é dividido em camadas e utiliza o padrão de projeto Fachada [18] como ponto de acesso a todas as funcionalidades do sistema. O sistema é divido nas camadas DATA, LOGIC, WEB e BEANS. Na implementação do sistema, essas camadas são representadas por pacotes. Na camada DATA estão implementadas as interfaces de acesso ao repositório de dados, bem como as classes que implementam essas interfaces. Na camada LOGIC estão todas as classes que recebem as requisições vindas da Fachada do sistema e implementam as funcionalidade de negócio e repassam as requisições e cadastro de registros para a camada DATA. Na camada WEB estão todos os *Servlets* que recebem as informações de entrada do usuário através de páginas escritas em JSP, e repassam essas informações pro controle da Fachada do sistema. No pacote BEANS estão todas as classes básicas do sistema que especificam as entidades que são manipuladas, sua estrutura interna e seu métodos de acesso (*gets* e *sets*).

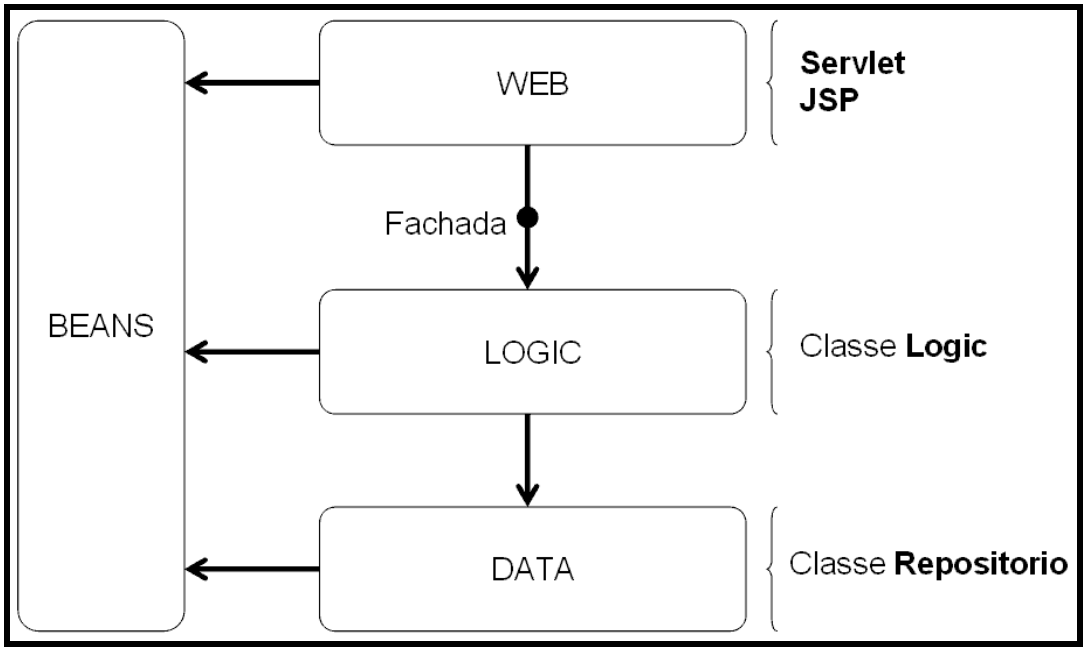

Figura 2. Arquitetura do Sistema de Cadastro de Casos de Uso

O sistema foi desenvolvido utilizando a plataforma Java, versão 5.0 [19]. Para dar suporte ao desenvolvimento foi utilizada a IDE (*Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento) Eclipse, versão 3.2 [20]. Como servidor de aplicação *web*, foi utilizado o Apache Tomcat [21], versão 5.5. O banco de dados utilizado foi o MySQL [22].

O cadastro dos casos de uso no Sistema de Cadastro de Casos de Uso é efetuado manualmente pelo usuário através do preenchimento de um formulário. O usuário insere no sistema a descrição do caso de uso, os atores que podem acessar a funcionalidade descrita no caso de uso, os passos para a execução e os fluxos alternativos e de exceção. Durante o cadastro dos passos para a execução do caso de uso o próprio usuário define o que é ação do usuário da funcionalidade e o que é resposta do sistema.

O projeto passou por diversas dificuldades e infelizmente não pode ser concluído integralmente dentro do escopo e prazo planejados. Os módulos de cadastro de campos e de geração de casos de teste não foram desenvolvidos. Inicialmente o objetivo do projeto era desenvolver um sistema de geração de casos de teste, entretanto, só foi possível desenvolver os módulos relacionados ao cadastro de casos de uso. Como dito anteriormente, o objetivo desta monografia é desenvolver o módulo de cadastro de casos de teste para o sistema de cadastro de caso de uso.

Capítulo 5

Geração Automática de Casos de Teste

Todo processo de elaboração de casos de teste de software necessita do uso de algum modelo para auxiliar na criação, seleção e verificação dos testes. Em geral, esses modelos são implícitos e utilizados de maneira inconsciente pelo projetista de teste. É como se existisse um modelo mental no cérebro do Testador descrevendo a forma como o sistema se comporta, permitindo que este compreenda as capacidades do software e teste de forma mais eficiente a grande variedade de possíveis comportamentos do sistema a ser testado [3].

Na literatura da Engenharia de Software existe uma grande abundância de estilos de teste. Vários desses estilos têm sido aplicados na indústria como soluções para garantir a crescente demanda por qualidade de software. Testes baseados em modelos (*Model-based Testing* ou MBT) é um termo geral que designa uma abordagem comum nas quais atividades de testes se baseiam, como geração de casos de teste e avaliação de resultados de teste, em um modelo de aplicação que está sendo ou será testada [23].

Nos últimos anos tem se expandido cada vez mais o uso de modelos explícitos para o desenvolvimento de software, como por exemplo, o uso de UML no processo de análise e projeto de aplicações orientadas a objeto [24]. O uso destes modelos no mercado de TI (Tecnologia da Informação) com propósito de gerar casos de testes ainda se encontra no início, embora uma parcela significativa das indústrias de telecomunicações, aeroespacial e de microeletrônicos já venhaexperimentando modelos para verificação e geração de testes durante a última década.

Na Seção 5.1 será descrita uma forma de elaborar casos de teste a partir de um caso de uso. Esse modelo foi utilizado como inspiração para a criação do algoritmo de geração automática de casos de teste descrito na Seção 5.2.

5.1 Elaboração de Casos de Teste

Uma das etapas de um processo de testes que tende a demandar um considerável tempo e esforço da equipe de testes é a atividade de projeto de testes. Não existem dados e pesquisas que apóiem essa afirmação, pois esta se trata de um senso comum e de uma situação já conhecida e enfrentada por testadores.

Os casos de testes são geralmente escritos manualmente. Esse tipo de atividade pode introduzir erros, falhas e redundâncias. Para reduzir esses problemas, é necessária uma série de revisões e validações dos casos de teste projetados. A geração automática de casos de teste e as técnicas de teste baseadas em modelos – *Model-Based Testing* – surgem como abordagens promissoras para minimizar o esforço na criação de casos de teste resultando em um maior controle da qualidade e na redução de custos inerentes do processo [3], [4].

Os casos de testes são elaborados de acordo com o objetivo dos testes. Para a elaboração de testes podem ser considerados os requisitos funcionais e os requisitos não-funcionais. Com requisitos não-funcionais, podem ser elaborados testes que verificam questões como segurança, desempenho, usabilidade e integridade dos dados. Neste trabalho, o foco é na elaboração de casos de teste que verificam os requisitos funcionais dos sistemas.

Em geral, os testes funcionais são escritos baseados em documentos que especificam o comportamento do sistema diante da interação com o usuário e quando submetidos a determinadas entradas. Os documentos utilizados para elaboração dos casos de testes podem ser os documentos de requisitos, os documentos de casos de uso ou até mesmo estórias da metodologia de desenvolvimento XP [16]. Neste trabalho, será abordada a elaboração de casos de teste baseada em documentos de casos de uso.

Na elaboração de casos de teste baseada em casos de uso são escritos roteiros de teste com os pré-requisitos para a execução, os passos e seus respectivos resultados esperados, bem como as entradas fornecidas e os dados de saída do sistema. Considere o caso de uso exibido na Tabela 1. O exemplo da tabela é retirado do livro "*Applying Use Cases: A Practical Guide*" [17]. Algumas pequenas modificações foram efetuadas no caso de uso para satisfazer as necessidades desse exemplo. Na Tabela 1 é exibido apenas o fluxo principal do caso de uso de um sistema hipotético de uma loja virtual.

Tabela 1. Caso de uso Fazer Pedido

CDU001 – Fazer Pedido
Atores: Cliente
Pré-condição:
O usuário deve ter feito *log-in* e obtido autorização do sistema
Fluxo Principal:
1. O caso de uso começa quando o cliente seleciona "fazer pedido". 2. O sistema exibe a Tela de Fazer Pedido 3. O cliente fornece seu nome e endereço. 4. O cliente fornece os códigos do produto. 5. O sistema devolve as descrições e o preço de cada produto. 6. O sistema calculará os valores totais para cada produto fornecido. 7. O cliente fornece as informações sobre cartão de crédito. 8. O cliente submete os dados ao sistema. 9. O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento. 10. Quando o pagamento é confirmado, o pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente.
Pós-condição:
O pedido deve ter sido gravado no sistema e marcado como confirmado.

De modo geral, um caso de uso possui um identificador e um nome. Também é comum relatar os atores do sistema que podem executar o caso de uso. Para um caso de uso ser executado é necessário se satisfazer um conjunto de pré-condições. Um caso de uso pode possuir diversos fluxos: normalmente possui um fluxo principal, também chamado de fluxo básico, que descreve o passo a passo para executar determinada funcionalidade do sistema com sucesso; fluxos secundários ou alternativos, que representam formas diferentes de interagir com o sistema para realizar a funcionalidade; fluxos de exceção ou erro, que descrevem como o sistema se comporta diante de uma ação que não corresponde à seqüência normal de interação do usuário com o sistema. Um caso de uso também descreve as pós-condições, que especificam o estado do sistema após a execução da funcionalidade. Um exemplo visual dos fluxos de um caso de uso é exibido na Figura 3. O fluxo principal é representado pela seta sólida, os fluxos alternativos são

representados pelas setas tracejadas e os fluxos de exceção são representados pelas setas pontilhadas.

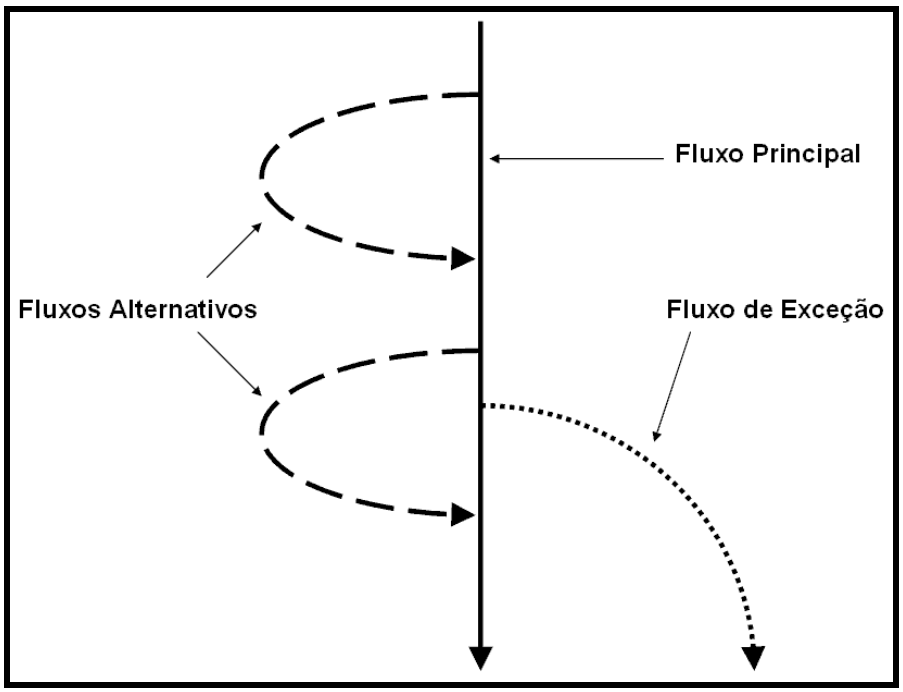

Figura 3. Um Caso de Uso e seus Fluxos

Para cada fluxo possível de um caso de uso, como fluxo principal ou básico, fluxos secundários ou alternativos e fluxos de exceção ou de erro, são elaborados casos de teste que exercitam esses fluxos com o intuito de verificar se estes foram corretamente implementados. Para casos de uso simples, com uma quantidade reduzida de fluxos alternativos e de exceção, essa atividade não demanda tanto tempo e esforço.

Entretanto, para casos de uso complexos, com vários fluxos alternativos, fluxos de exceção e fluxos que surgem a partir de outros fluxos alternativos, essa atividade se torna exaustiva e muito complexa demandando grande atenção por parte do projetista e uma revisão rigorosa para garantir a elaboração de pelo menos um caso de teste para cada caminho possível. Um exemplo visual de um caso de uso complexo pode ser visto na Figura 4. Mais uma vez o fluxo principal é representado pela seta sólida, os fluxos alternativos são representados pelas setas tracejadas e os fluxos de exceção são representados pelas setas pontilhadas.

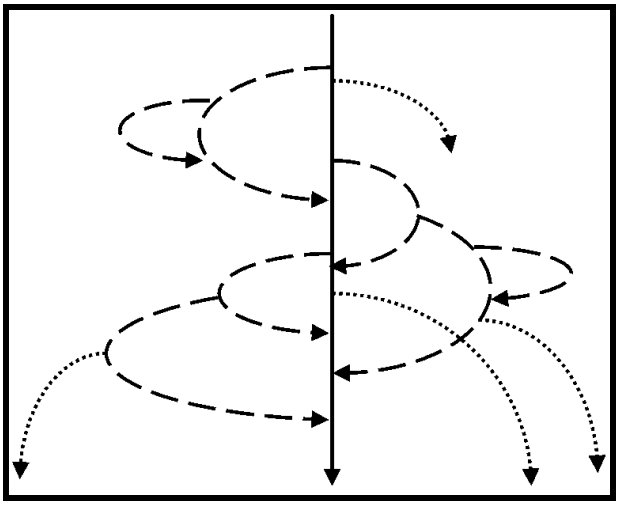

Figura 4. Exemplo de Caso de Uso Complexo

Um exemplo de caso de teste que verifica se o sistema funciona como especificado no Fluxo Principal do caso de uso Fazer Pedido pode ser visto na Tabela 2. Assim como um caso de uso, um caso de teste possui um identificador e um nome. Como dito anteriormente, um caso de teste possui um roteiro de execução de teste. Para se executar um caso de teste, primeiramente é necessário satisfazer todos os pré-requisitos. Em seguida, o testador executa os passos e verifica se o sistema se comporta de acordo com o esperado observando a coluna de Resultados esperados após a execução de cada passo. Caso o sistema se comporte como o esperado o caso de teste "Passa", ou seja, o sistema se comporta da forma como foi especificado. Caso o resultado esperado de algum passo esteja diferente da resposta do sistema, o caso de teste "Falha", pois o sistema não se comporta de acordo com suas especificações.

Tabela 2. Exemplo de Caso de Teste

CT001 – Fazer Pedido com sucesso	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça nome e endereço	
3. Forneça os códigos do produto	O sistema devolve as descrições e o preço de cada produto e calcula os valores totais para cada produto fornecido
4. Forneça as informações sobre cartão de crédito	
5. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
6. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

É diante desse cenário de escrita de casos de teste para casos de uso complexos que a abordagem de geração automática surge como alternativa para redução do esforço do projeto e garantia de que todos os fluxos do caso de uso estão sendo testados. Na seção a seguir descreveremos um algoritmo de geração automática de casos de teste baseado no modelo apresentado anteriormente.

5.2 Algoritmo de Geração Automática de Casos de Teste

Nesta seção será descrita a idéia geral do mecanismo de geração automática de casos de teste implementado em alto nível. Como dito na seção anterior, casos de teste são gerados a partir de modelos mentais que representam o comportamento do sistema. Em geral esses modelos mentais são criados de forma *ad hoc* pelo projetista de teste. Para criação desses modelos, o projetista se baseia nas especificações de um sistema. Como dito na Seção 5.1, essas especificações podem ser documentos de casos de uso, estórias ou qualquer outra forma de especificação.

Parte das ferramentas de geração automática de casos de teste citadas no Capítulo 2, Seção 2.2 se baseiam em modelos descritos em linguagens de especificação formal. Uma outra parcela desses sistemas trabalha com UML como linguagem de modelagem de sistema para geração de casos de teste. No caso do sistema desenvolvido neste trabalho, o modelo utilizado como entrada para a ferramenta de geração automática de casos de teste é o documento de casos de uso. Entretanto é importante que fique claro que o sistema não receberá como entrada documentos de casos de uso escritos em programa de edição de texto. Os casos de uso usados como entrada serão os casos de uso cadastrados manualmente no Sistema de Cadastro de Casos de Uso.

Como visto na Seção 5.1, um caso de uso é formado por um fluxo principal e diversos fluxos alternativos ou de exceção que partem do fluxo principal. Tendo em mente o modelo visual dos fluxos de um caso de uso da Figura 3, são construídos casos de teste para cada fluxo. Na Figura 5 são exibidos alguns exemplo de possíveis casos de testes gerados para o modelo visual de um caso de uso da Figura 3.

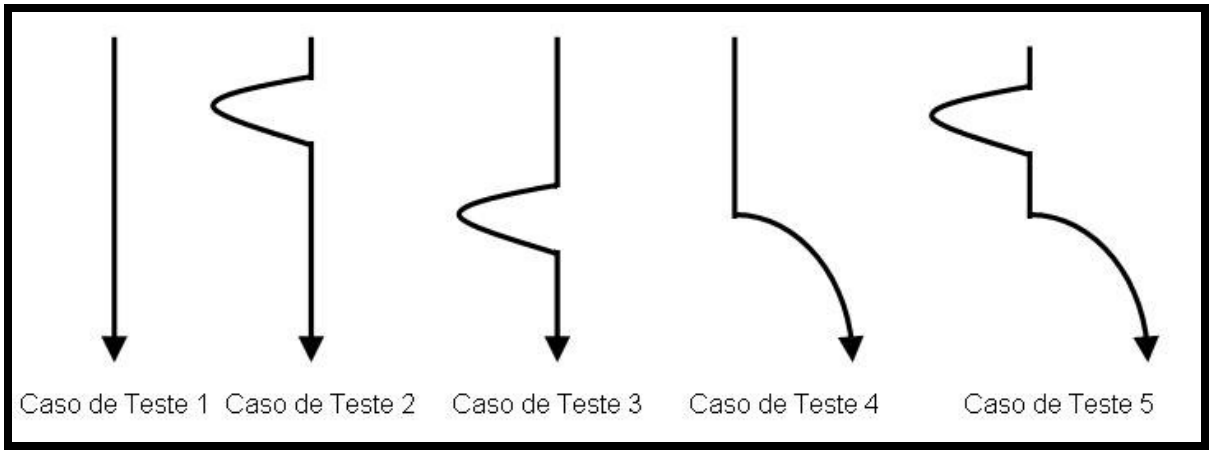

Figura 5. Possíveis Casos de Teste

O trabalho do projetista de teste, quando se trata de verificar as funcionalidades do sistema, é elaborar pelo um menos um teste para cada caminho possível de um caso de uso. Quando se trata de um caso de uso simples, como o da Figura 3, essa atividade não é tão complexa, apesar de demandar uma quantidade de tempo considerável. Entretanto, quando se trata de um caso de uso complexo, como o da Figura 4, o tempo e o custo necessário se tornam muito maiores.

A idéia do algoritmo para geração automática de casos de teste é gerar todos os fluxos possíveis, garantindo que nenhum fluxo será esquecido e proporcionando uma grande economia de tempo e esforço.

Para gerar os casos de teste, o algoritmo foi dividido em três etapas descritas a seguir:

1ª Etapa – Geração dos fluxos:

- Para cada fluxo de um caso de uso cadastrado, seja ele principal, alternativo ou de exceção, é gerado um caso de teste;

2ª Etapa – Determinação dos caminhos:

- O Fluxo Principal é percorrido para verificar onde surgem fluxos alternativos e de exceção;
- Cada Fluxo alternativo e de exceção é percorrido para verificar se surgem fluxos a partir destes;
- Uma estrutura composta por listas é criada com os passos do fluxo principal concatenados com os passos de cada fluxo alternativo, sendo cada lista da estrutura um caminho possível do caso de uso;

3ª Etapa – Geração de casos de teste:

- Cada lista da estrutura é percorrida e os passos dos casos de teste vão sendo gerados a partir da ligação dos casos de teste gerados na 1ª Etapa;

Para facilitar a compreensão do funcionamento do algoritmo, conduziremos um exemplo. Primeiramente, considere o exemplo da Tabela 1. Neste exemplo é exibido apenas o fluxo principal, que representa o caso básico de sucesso. No entanto existem caminhos alternativos para realização desse caso de uso. Um identificador deve ser atribuído para cada fluxo alternativo. Como identificador adotaremos a sigla FA para designar um fluxo alternativo seguido dos dígitos referentes ao caso de uso que o fluxo alternativo está associado. Por exemplo, no caso de um fluxo alternativo para o CDU001, o seu identificador terá o prefixo FA001. Além desse prefixo, diferenciaremos os fluxos alternativos utilizando dígitos identificadores. Por exemplo, o primeiro fluxo alternativo do CDU001 terá como identificador FA001.01, o segundo será FA001.02 e assim por diante.

Prosseguindo com a especificação do CDU001, no passo 3 do fluxo principal, onde o cliente fornece seu nome e endereço, existe um fluxo alternativo que permite que o usuário forneça como entrada apenas o CEP. A descrição completa do fluxo alternativo é exibida na Tabela 3.

Tabela 3. Fluxo alternativo Fornece CEP

FA001.01 – Fornecer CEP
1. No passo 3 do fluxo principal, o cliente fornece apenas o CEP. 2. O sistema coloca automaticamente a cidade e o estado. 3. O sistema retorna ao passo 4 do fluxo principal.

Também é permitido ao cliente realizar o pedido de outros produtos, ou seja, existe outro fluxo alternativo. Esse fluxo parte do passo 4 do fluxo principal e retorna para o mesmo passo enquanto o cliente continuar adicionando produtos. A descrição desse fluxo pode ser vista na Tabela 4.

Tabela 4. Fluxo alternativo Adicionar Item

FA001.02 – Adicionar Item
1. No passo 4 do fluxo principal, o cliente pode selecionar “Adicionar Item”. 2. O sistema exibe o campo para adição do código de um produto. 3. O cliente fornece o código do produto. 4. O sistema fornece a descrição e o preço do produto. 5. O sistema atualiza o valor total. 6. O sistema retorna para o passo 4 do fluxo principal.

Além dos fluxos alternativos descritos anteriormente, o caso de uso também possui um fluxo de exceção. Assim como no caso dos fluxos alternativos, também é necessário utilizar

identificadores para os fluxos de exceção. Nesse caso a regra é a mesma dos fluxos alternativos, com a única exceção de que a sigla será FE e não FA.

Voltando a descrição do fluxo de exceção, no passo 7 do fluxo principal, após a submissão dos dados preenchidos pelo cliente, caso alguma informação esteja incorreta o sistema retorna uma mensagem de erro. A descrição desse fluxo de exceção é mostrada na Tabela 5.

Tabela 5. Fluxo de exceção Informação Incorreta

FE001.01 – Informação incorreta
1. No passo 7 do fluxo principal, se alguma informação estiver correta, o sistema exibe uma mensagem de erro “Informações Incorretas”.

A especificação completa do caso de uso CDU001 - Fazer Pedido é composta pelo fluxo principal (Tabela 1), fluxos alternativos (Tabela 3 e Tabela 4) e pelo fluxo de Exceção (Tabela 5). Quando esse caso de uso é selecionado e utilizado como entrada para o algoritmo de geração de casos de teste apresentado no início dessa seção, ele passa pelas três etapas de geração.

Na primeira etapa, chamada Geração dos Fluxos, são criados casos de teste para cada fluxo. O caso de teste para o fluxo principal é exibido na Tabela 2. Para os fluxos alternativos FA001.01 – Fornecer CEP e FA002.02 – Adicionar Item são criados os casos de teste exibidos na Tabela 6 e Tabela 7 respectivamente. E para o fluxo de exceção FE001.01 – Informação incorreta é gerado o caso de teste da Tabela 8.

Tabela 6. Fluxo de Fornecer CEP

Fornecer CEP	
Pré-requisitos:	
Passos	Resultados Esperados
1. Forneça o CEP	O sistema coloca automaticamente a cidade e o estado

Tabela 7. Fluxo de Adicionar Item

Adicionar Item	
Pré-requisitos:	
Passos	Resultados Esperados
1. Selecione “Adicionar Item”	O sistema exibe o campo para adição do código de um produto
2. Forneça o código do produto	O sistema fornece a descrição e o preço do produto e atualiza o valor total

Tabela 8. Fluxo de Informação Incorreta

Informação Incorreta	
Pré-requisitos:	
Passos	Resultados Esperados
1. Digite uma informação incorreta	O sistema exibe uma mensagem de erro "Informações Incorretas"

Na segunda etapa da geração, chamada de Determinação dos caminhos, uma estrutura é criada percorrendo cada fluxo para determinar onde surgem fluxos alternativos ou de exceção e pra onde estes fluxos retornam, ou seja, para que passo de que outro fluxo eles retornam. No caso do exemplo estudado, existem fluxos partindo e retornando apenas de e para o fluxo principal. A estrutura é composta por listas, onde cada lista se refere a um caminho possível.

Partindo do Fluxo Principal para o fluxo alternativo FA001.01, temos a lista de passos abaixo:

- [CDU001-1; CDU001-2; FA001.01-1; FA001.01-2; FA001.01-3; CDU001-4; CDU001-5; CDU001-6; CDU001-7; CDU001-8; CDU001-9; CDU001-10]

Do Fluxo Principal para o fluxo alternativo FA001.02 temos a lista:

- [CDU001-1; CDU001-2; CDU001-3; CDU001-4; FA001.02-1; FA001.02-2; FA001.02-3; FA001.02-4; FA001.02-5; FA001.02-6; CDU001-4; CDU001-5; CDU001-6; CDU001-7; CDU001-8; CDU001-9; CDU001-10]

Do Fluxo Principal para o fluxo de exceção temos:

- [CDU001-1; CDU001-2; CDU001-3; CDU001-4; CDU001-5; CDU001-6; CDU001-7; FE001.01-1]

Após a criação da estrutura de dados com os caminhos possíveis, a segunda etapa do algoritmo chega ao fim. Na terceira etapa, denominada Geração de Casos de Teste, a estrutura será percorrida e os casos de teste gerados na primeira etapa serão concatenados. O caso de teste que verifica o caso de sucesso do caso de uso, ou seja, o caso de teste do fluxo principal gerado na primeira etapa não é modificado. Os demais casos de testes são alterados. A saída da terceira e última etapa, e consequentemente a saída do algoritmo, são os casos de teste das tabelas: Tabela 9, Tabela 10, Tabela 11 e Tabela 12.

Tabela 9. Caso de Teste Fazer Pedido com Sucesso

CT001 – Fazer Pedido com sucesso	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça nome e endereço	
3. Forneça os códigos do produto	O sistema devolve as descrições e o preço de cada produto e calcula os valores totais para cada produto fornecido
4. Forneça as informações sobre cartão de crédito	
5. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
6. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

Tabela 10. Caso de Teste Fornecer CEP

CT002 – Fornecer CEP	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça nome e endereço	
3. Forneça o CEP	O sistema coloca automaticamente a cidade e o estado
4. Forneça os códigos do produto	O sistema devolve as descrições e o preço de cada produto e calcula os valores totais para cada produto fornecido
5. Forneça as informações sobre cartão de crédito	
6. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
7. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

Tabela 11. Caso de Teste Adicionar Item

CT003 – Adicionar Item	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça nome e endereço	
3. Forneça os códigos do produto	O sistema devolve as descrições e o preço de cada produto e calcula os valores totais para cada produto fornecido
4. Selecione “Adicionar Item”	O sistema exibe o campo para adição do código de um produto
5. Forneça o código do produto	O sistema fornece a descrição e o preço do produto e atualiza o valor total
6. Forneça as informações sobre cartão de crédito	
7. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
8. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

Tabela 12. Caso de Teste Informação Incorreta

CT004 – Informação Incorreta	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça nome e endereço	
3. Forneça os códigos do produto	O sistema devolve as descrições e o preço de cada produto e calcula os valores totais para cada produto fornecido
4. Forneça as informações sobre cartão de crédito	
5. Digite uma informação incorreta	O sistema exibe uma mensagem de erro “Informações Incorretas”

No Capítulo 6 demonstraremos um experimento conduzido com o objetivo de comparar a abordagem manual de geração de casos de teste com a abordagem de geração automática proposta por este trabalho.

Capítulo 6

Análise e Comparação dos Resultados

Neste capítulo será apresentado um experimento comparativo com o intuito de validar a utilização do algoritmo de geração automática de casos de teste proposto por este trabalho. O experimento consistirá na execução de um projeto de testes baseado em um documento de casos de uso. Serão realizados dois projetos de teste para um mesmo sistema que será escolhido como estudo de caso. O projeto de testes será realizado de forma manual, que é a maneira mais utilizada atualmente e de forma automática, através da utilização do algoritmo de geração de casos de teste.

Durante a execução dos projetos, algumas métricas serão capturadas e utilizadas como meio de comparação entre as duas abordagens aplicadas, a manual e a automática. Como dito anteriormente, um sistema será utilizado como exemplo para execução do projeto de testes. As métricas utilizadas como parâmetro de comparação e os motivos destas serem escolhidas serão expostos na Seção 6.1. Também na Seção 6.1, será explicada a metodologia utilizada para execução do experimento tanto na abordagem manual como na automática e serão fornecidas maiores informações sobre o sistema utilizado no estudo e seus casos de uso.

Na Seção 6.2 serão mostrados os resultados obtidos para cada experimento. Os resultados também serão comparados e discutidos e serão avaliadas as vantagens de cada abordagem, bem como seus pontos fracos, verificando principalmente se a abordagem de geração automática de casos de teste proposta por esse trabalho realmente representa um ganho significativo no desempenho da atividade de projeto de testes.

É importante ressaltar que com a condução de apenas um estudo comparativo, considerando apenas um sistema, não é o suficiente para efetuar uma validação apurada da abordagem automática. Para tanto seria necessária a realização de diversos experimentos,

considerando os mais variados sistemas, com diferentes complexidades de casos de uso, e quantidades de casos de uso. Apenas com um estudo mais rigoroso, se teria uma idéia melhor de onde cada abordagem pode ser mais apropriada. Portanto, a análise comparativa realizada nesse trabalho se constitui apenas como uma verificação preliminar da abordagem proposta comparada com o modelo de elaboração manual de casos de teste do qual ela foi inspirada.

6.1 Metodologia

Com o objetivo de verificar e validar a ferramenta de geração de casos de teste proposta, serão conduzidos dois experimentos de avaliação. Como já adiantado anteriormente, esses experimentos consistem em um projeto de testes para um sistema baseado em seus casos de uso. Para os projetos de testes das duas abordagens será utilizado o mesmo sistema. É importante ressaltar que nesses experimentos estaremos considerando apenas os testes funcionais do sistema, ou seja, não serão elaborados testes de segurança, usabilidade, desempenho, carga, *stress* ou qualquer outra categoria de testes.

No experimento número 1, será utilizada a abordagem manual de elaboração de casos de teste. A técnica utilizada na elaboração dos casos de teste será a mesma utilizada anteriormente na Seção 5.1. A partir dos documentos de casos de uso com os seus fluxos principais, alternativos e de exceção serão criados os casos de teste. Para este experimento, assim como no exemplo mostrado na Seção 5.1, não serão considerados os dados de entradas para os testes nem os campos necessários para entrada do usuário e os campos de saída do sistema. O foco principal é verificar apenas a questão da elaboração de testes para cada fluxo. Além disso, o objetivo é simplificar o experimento.

No experimento número 2, o projeto de testes será executado utilizando a abordagem automática proposta na Seção 5.2. Assim como no experimento anterior, não serão considerados os campos de entrada do usuário e os campos de saída do sistema, bem como os valores de entrada.

Durante a execução e ao final dos experimentos algumas métricas serão capturadas para comparação das duas abordagens. Essas métricas serão tanto quantitativas quanto qualitativas. As métricas avaliadas serão:

1. Esforço do projeto de testes;
2. Número de casos de teste gerados;
3. Número de casos de teste com possíveis registros duplicados;

4. Qualidade dos testes;
5. Cobertura dos testes;
6. Impacto no processo de testes.

As métricas acima serão detalhadas e discutidas ponto a ponto a seguir bem como as razões de terem sido escolhidas como critérios de avaliação e comparação entre as abordagens manual e automática.

Consideramos como Esforço do projeto de testes, a relação entre o tempo necessário para o projeto ser realizado e a quantidade de homens empregados nessa atividade. Os dois experimentos foram realizados por apenas um homem, portanto a variável a ser considerada será o tempo. O tempo dos dois projetos será cronometrado desde a análise do documento de casos de uso, a elaboração dos testes e a inspeção dos casos de teste projetados.

Em relação ao Número de casos de teste gerados, ao final dos projetos de testes, obteremos a quantidade de testes gerada por cada abordagem. Isoladamente, o Número de casos de teste gerados por cada abordagem não demonstra claramente qual delas é a mais eficiente. É necessário utilizar outros critérios de avaliação para verificar se todos os testes gerados são realmente relevantes ou até mesmo se são válidos. Uma quantidade maior de casos de teste por si só não prova que uma abordagem é superior à outra. Não adianta gerar uma quantidade enorme de casos de teste e aumentar o esforço durante a execução, se estes casos de teste não têm relevância ou potencial para capturar novos erros.

Ao final dos projetos, os casos de teste gerados por cada abordagem serão avaliados com o intuito de verificar o Número de casos de teste com possíveis registros duplicados. Testes com possíveis registros duplicados são testes que têm grande probabilidade de capturar o mesmo erro por repetir passos que são executados em outros testes, o que pode a levar a geração de registros duplicados e retrabalho. É importante não confundir esse conceito com a idéia de testes repetidos e que possuem o mesmo objetivo, ou seja, casos de teste redundantes.

Os casos de testes gerados por cada abordagem serão avaliados para verificar a forma como foram escritos, a sua clareza e a facilidade de compreensão dos testes para a atividade de execução. A Qualidade dos testes será avaliada para comparar quais das abordagens produzem testes com maior probabilidade de serem compreendidos e de ajudar o testador na etapa de execução de testes.

É importante avaliar, ao final do projeto de testes, se os casos de teste projetados cobrem todos os caminhos possíveis de execução de uma funcionalidade. Com essa premissa em mente,

ao final dos projetos, será observada a Cobertura dos testes de cada abordagem para avaliar se estas são capazes de avaliar uma funcionalidade como um todo.

Por fim, será verificado o Impacto no processo de testes da utilização da abordagem de geração automática utilizando o algoritmo proposto por este trabalho. Serão verificados os ganhos da utilização desta abordagem e as vantagens e desvantagens de sua adoção em um processo de testes hipotético.

No projeto utilizando a abordagem de elaboração manual de casos de teste, primeiramente serão analisados os casos de uso do sistema. Cada caso de uso será analisado separadamente e casos de testes serão gerados para cada fluxo, seja principal, alternativo ou de exceção. Após a elaboração de todos os casos de testes, será realizada uma etapa de inspeção, onde os casos de testes serão avaliados, aprovados ou corrigidos. Na etapa de inspeção, também é possível que novos casos de testes sejam criados, caso algum fluxo não tenha sido coberto ou seja verificada a possibilidade de criação de novos testes. Não serão criados casos de teste que verificam a integração entre casos de uso.

Utilizando a abordagem de geração automática de casos de teste proposta por este trabalho, também será considerada a etapa de análise dos casos de uso do sistema. Também será contabilizado no tempo, o esforço para o cadastro dos casos de uso no Sistema de Cadastro de Casos de Uso. O tempo que a ferramenta necessita para gerar os casos de teste também fará parte do esforço total do projeto. Assim como na abordagem manual, os testes também serão inspecionados.

O sistema utilizado nos projetos de testes será uma aplicação de loja virtual, já apresentada anteriormente no Capítulo 5. A aplicação de loja virtual é um sistema simples, composto apenas por sete casos de uso. Um sistema simples foi escolhido devido o reduzido tempo para a realização dos experimentos. Além desse motivo, esse exemplo foi escolhido por se tratar de uma especificação pronta, já utilizada na literatura [17]. Além do caso de uso Fazer Pedido, realizaremos o projeto de testes para os casos de uso Verificar Pedido, Cancelar Pedido, Fornecer Produto, Procurar Pedido, Produto em Oferta e Cliente Especial.

6.2 Resultados Obtidos

Nesta seção apresentaremos os resultados obtidos durantes os experimentos e discutiremos quais das abordagens tiveram melhor desempenho neste projeto de testes específico em relação a cada critério avaliado. Ao final da Seção, discutiremos qual abordagem teve maior sucesso

considerando todas as métricas de comparação e em quais situações uma abordagem pode ter um melhor desempenho que a outra.

6.2.1 Esforço do projeto de testes

Durante o projeto de testes foi cronometrado o tempo necessário para elaboração de casos de teste. O objetivo de capturar essa métrica é avaliar se o algoritmo proposto contribui com uma diminuição no esforço da atividade de projetar testes. Obtendo uma redução significativa do tempo, consequentemente teremos uma redução nos custos do processo de testes.

A Tabela 13 apresenta de forma detalhada o tempo gasto em cada atividade executada no projeto de testes utilizando a abordagem manual e o tempo total gasto. A Tabela 14 apresenta os resultados obtidos utilizando a abordagem automática proposta neste trabalho.

Tabela 13. Abordagem Manual

Atividades da Geração Manual de Casos de Teste	**Esforço (h)**
Análise	00:14:27
Elaboração de Testes	01:33:58
Inspeção dos casos de teste	01:10:42
Esforço Total da Abordagem Manual	02:59:07

Tabela 14. Abordagem Automática

Atividades da Geração Automática de Casos de Teste	**Esforço (h)**
Análise	00:15:49
Cadastro do documento de Casos de Uso	00:37:38
Geração Automática dos Casos de Teste	00:00:10
Inspeção dos casos de teste	00:20:51
Esforço Total da Abordagem Manual	01:14:28

A partir dos dados obtidos experimentalmente, podemos observar que a diferença entre os tempos de análise é irrelevante. A atividade de análise das duas abordagens é semelhante e não tem como representar ganhos ou vantagens para nenhuma das duas. Em relação à abordagem automática, as atividades de Cadastro do documento de Casos de Uso e de Geração Automática dos Casos de Teste correspondem à atividade de Elaboração de Testes da abordagem manual. A

soma destas duas atividades da abordagem automática obteve uma redução de cerca de 58% de esforço em relação à atividade de Elaboração de Testes da abordagem manual.

A atividade de Inspeção dos casos de teste da abordagem manual teve um gasto de tempo elevado, contabilizando cerca de 33% do esforço total do projeto de testes. Após a etapa de Elaboração de Testes, 26 testes foram criados. Na etapa de Inspeção, 6 casos de teste foram adicionados e 15 testes foram corrigidos. Em uma situação ideal, os casos de testes elaborados devem ser inspecionados por uma pessoa diferente da que projetou os testes, visto que o projetista tem menos probabilidade de encontrar erros. Entretanto, neste trabalho essa boa prática não pôde ser utilizada.

A etapa de Geração Automática de Casos de Teste foi bastante curta, como o esperado. Nesta etapa foram gerados 22 testes. Esse número certamente seria superior se o algoritmo de geração automática de casos de testes estivesse completamente implementado.

A atividade de Cadastro dos Casos de Uso da abordagem automática correspondeu a cerca de 40% da atividade de elaboração de casos de teste da abordagem manual. Entretanto esse número poderia ser reduzido se o cadastro dos casos de uso na ferramenta já fizesse parte do processo de análise e desenvolvimento adotado. Outra forma de reduzir esse tempo seria permitir que o sistema recebesse como entrada um documento de casos de em um programa de edição de texto e gerasse os casos de teste.

A atividade de Inspeção dos Casos de Teste da abordagem automática demandou muito menos tempo do que a da abordagem manual, totalizando uma redução de 70%. Na abordagem automática não foram adicionados novos testes e os testes não precisaram ser corrigidos. Além da atividade normal de inspeção para verificar se testes foram criados para cada caminho possível, foi gasto mais tempo para completar os testes referenciando outros testes como pré-requisitos. Esse tipo de atividade não seria necessário se todas as etapas do algoritmo tivessem sido implementadas. Entretanto, apesar dessa atividade extra, a abordagem automática reduziu em cerca de 58% o tempo total do projeto de testes.

Antes de finalizar essa seção é importante ressaltar dois pontos. Primeiramente, devido a limitações de tempo não foi possível executar o experimento com um caso de uso mais complexo e apesar da diferença significativa entre as duas abordagens, o ganho da abordagem tende a ser ainda maior quando existem mais fluxos alternativos e de exceção. E por fim, ainda é necessário avaliar o ganho real da abordagem automática conduzindo experimentos com diferentes casos de uso de diferentes complexidades.

6.2.2 Número de casos de teste gerados

Outra métrica importante na avaliação é o número de casos de teste gerados por cada abordagem. Como dito na Seção 6.1, essa métrica isolada não possui relevância. Ela é importante para verificar se casos de testes para todos os caminhos são gerados, e avaliar se todos os casos de testes gerados são realmente relevantes e possuem potencial de capturar diferentes falhas.

A Tabela 15 apresenta o número de casos de teste gerados por cada abordagem.

Tabela 15. Número de Testes Gerados

Técnica Utilizada	**Total de Casos de Teste**
Abordagem Manual	32
Abordagem Automática	22

Ao final do projeto de testes utilizando a abordagem manual, foram gerados 32 casos de teste. Utilizando a abordagem automática, foram gerados 22 testes, 10 a menos do que a abordagem manual. Isso ocorreu porque os testes foram gerados automaticamente utilizando apenas a primeira etapa do algoritmo proposto. No caso da implementação integral do algoritmo, provavelmente a quantidade de testes gerados seria igual ou maior do que a da abordagem manual.

Uma questão importante a ser comentada em relação à abordagem manual é que esta permite a criação de testes para verificar situações não previstas nos casos de uso. Devido à criatividade e experiência do testador é possível verificar situações possíveis de ocorrer no sistema que não foram previstas na sua especificação. Na abordagem manual foram criados 2 casos de testes que não estavam previstos no Caso de uso. Esses tipos de casos de teste têm grande potencial de encontrar falhas. Além desses 2 testes, mais 1 teste foi criado devido à experiência do projetista. Devido a sua estrutura, esse último teste é difícil de ser gerado automaticamente.

Na abordagem automática proposta esse tipo de criatividade ainda não é possível. Entretanto isso não invalida a geração automática de casos de teste, pois o objetivo dessa abordagem não é substituir o papel do projetista, e sim auxiliar e tornar o processo de testes mais ágil. A criatividade e experiência do projetista podem ser empregadas durante a atividade de inspeção, onde novos casos de teste podem ser adicionados e alterados para satisfazer situações não cobertas pelos casos de teste gerados inicialmente.

Apesar de gerar menos testes, a abordagem automática foi capaz de gerar 4 testes de 4 caminhos que não foram elaborados manualmente. Apesar da Inspeção dos testes elaborados

manualmente, testes que verificassem esses caminhos não foram criados. Além destes 4 testes extras, os testes gerados automaticamente exercitam 9 caminhos válidos do requisito que não são verificados pelos testes elaborados manualmente. Essa é mais uma vantagem da abordagem automática proposta. Ela garante que todos os caminhos especificados no caso de uso sejam exercitados.

6.2.3 Número de casos de teste com possíveis registros duplicados

O número de casos de teste com possíveis registros duplicados é importante para avaliar a qualidade dos testes gerados e verificar se os testes gerados têm grande capacidade de capturar erros. Testes que repetem alguns passos realizados por outros testes têm grande probabilidade de capturar um erro encontrado anteriormente, entretanto é necessário que alguns passos que já foram realizados tenham de ser executados novamente com o objetivo de se verificar uma nova condição ou seguir por um outro caminho de execução da funcionalidade. No entanto, é importante verificar se os objetivos principais dos testes, ou seja, a principal funcionalidade ou o principal caminho a ser testado estão devidamente claros e diferenciados.

O número de testes com possíveis registros duplicados encontrados para cada abordagem é exibido na Tabela 16.

Tabela 16. Número de Testes com possíveis registros duplicados

Técnica Utilizada	**Total de Casos de Teste com possíveis registros duplicados**
Abordagem Manual	13
Abordagem Automática	3

É interessante comentar que a abordagem automática não teve um grande número de casos de teste com potencial para gerar o mesmo erro. Isso ocorreu porque o algoritmo de geração automática foi implementado apenas até a primeira etapa. Também é importante destacar que essa métrica não tem um impacto tão negativo quanto as outras, pois depende muito da maturidade da equipe de teste e de sua capacidade de comunicação para que registros de erro duplicados não sejam reportados. No entanto é importante ficar atento a esses testes que possuem passos repetidos, principalmente quando diferentes testadores estão executando testes do mesmo caso de uso.

6.2.4 Qualidade dos testes

Em relação à qualidade dos testes, os testes da abordagem manual são mais facilmente compreendidos pelos testadores. Isso ocorreu porque não foi possível implementar todas as etapas do algoritmo de geração automática proposto.

Os testes gerados automaticamente dependem muito da execução de testes anteriores. Apesar disso a geração dos casos de testes dessa forma proporcionam algumas vantagens como por exemplo, os testes gerados possuem menos passos, os testes podem ser reusados como pré-requisitos de outros testes, a manutenção dos testes se torna mais fácil, e caso algum teste que seja pré-requisito de outro falhe, este último pode ser bloqueado, diminuindo a probabilidade de reportar erros duplicados. A maior desvantagem dessa forma de geração é que ela dificulta o trabalho do executor dos testes, pois ele necessita buscar outros testes para efetuar a execução. Se o algoritmo tivesse sido completamente implementado, esse problema não ocorreria. Os testes seriam mais semelhantes aos testes gerados manualmente.

Outro problema da geração automática dos casos de teste é que os passos dos casos de teste são idênticos aos passos do caso de uso. Para os testadores é mais fácil compreender os testes quando os passos estão escritos no imperativo (ex.: “Clique”, “Selecione”, “Forneça”, etc.). Em geral os casos de uso se referem às ações executadas pelo usuário do sistema, e os passos são especificados na terceira pessoa (ex.: “O usuário clica”, “O usuário seleciona”, “O usuário fornece”, etc.). Compreendendo mais rapidamente os testes, a execução consequentemente se torna mais eficiente.

6.2.5 Cobertura dos testes

Em relação à cobertura de testes, a abordagem manual não cobriu 11 caminhos possíveis de execução das funcionalidades do requisito projetado. Durante a etapa de Inspeção foram adicionados 6 testes que não foram criados na atividade de Elaboração de Testes. Apesar disso, a inspeção não foi capaz de elaborar testes para estes 11 caminhos possíveis. Essa é uma desvantagem gravíssima da abordagem manual, pois é difícil apenas com a leitura do documento projetar testes para todos os caminhos e, dependendo da complexidade do caso de uso, o esforço necessário para elaborar todos os testes pode ser tão alto que pode inviabilizar o projeto de testes.

A abordagem automática, embora com menos casos de teste, foi capaz de cobrir todos os caminhos possíveis. Essa é a maior vantagem da abordagem automática em relação à abordagem manual. Apesar de gerar testes para todos os caminhos, para algumas situações específicas como

particularidades dos casos de uso e testes criados pela experiência do projetista, relatadas na Seção 6.2.2, a abordagem automática não foi capaz de gerar testes.

6.2.6 Impacto no processo de testes

Após a execução dos experimentos, é possível afirmar que a utilização da abordagem automática representa um impacto positivo no processo de testes. Em termos de aprendizagem para o projetista, o esforço é mínimo, visto que o cadastro dos casos de uso no Sistema de Cadastro de Caso de Uso pode ser aprendido facilmente.

A atividade de cadastro de casos de uso do sistema ainda pode ser minimizada se o Sistema de Cadastro de Caso de Uso fosse adotado como ferramenta de suporte ao processo de Análise e Projeto de Software, sendo necessário apenas que a equipe de testes gerasse os casos de teste utilizando o sistema e realizasse a atividade de inspeção para avaliar e corrigir os testes gerados ou projetar testes que demandam a experiência e a criatividade do projetista e não são possíveis de serem gerados automaticamente.

Em relação à atividade de Inspeção dos casos de teste gerados, esta é praticamente a mesma da inspeção realizada na abordagem manual. A única diferença é a necessidade de referenciar os passos de outros testes nos pré-requisitos de alguns testes, pois o Sistema de Cadastro de Casos de Uso ainda não é capaz de associar um fluxo alternativo ou de exceção a um passo de um caso de uso. Essa limitação impactou na implementação do algoritmo. Com a implementação dessa associação, a atividade de inspeção se tornará ainda mais semelhante à inspeção dos casos de teste manuais.

6.2.7 Considerações Finais

Como exposto anteriormente neste Capítulo, os experimentos foram executados apenas para um requisito de um sistema, devido às limitações de tempo. O ideal seria avaliar a abordagem automática diante de diversos requisitos de diferentes funcionalidades para observar como esta abordagem se comporta diante de várias situações.

Entretanto, a partir desses experimentos preliminares é possível concluir que o ganho da utilização da abordagem automática pode se tornar ainda maior quanto mais complexo for o requisito a ser testado. Esses experimentos indicam que quanto mais complexo e com mais fluxos um caso de uso, maior a redução do esforço da abordagem automática em relação à abordagem manual.

A redução do esforço do projeto de testes utilizando a abordagem automática foi de quase 60% em relação à abordagem manual. Essa redução é bastante significativa em termos do projeto como um todo. Além disso, a abordagem automática também resultou numa significativa redução no esforço da atividade de inspeção dos casos de teste.

A abordagem automática gerou menos testes que a manual, entretanto os testes gerados são capazes de verificar todos os fluxos possíveis do requisito, o que não ocorreu na abordagem manual, mesmo após a inspeção que devia garantir essa verificação. Quanto mais fluxos um caso de uso possui, mais difícil se torna o projeto de testes manual.

É importante destacar que a abordagem manual possibilita uma maior liberdade para o projetista de teste permitindo que este observe situações possíveis de ocorrer no sistema que não foram devidamente especificadas nos documentos de casos de uso. Já a abordagem automática está limitada a refletir exatamente o que foi especificado, não prevendo situações que podem causar potenciais falhas no sistema. Entretanto, a criatividade do projetista não é totalmente eliminada se for utilizada a abordagem automática como suporte. O projetista de testes ainda possui a liberdade de avaliar os testes gerados automaticamente e modificá-los ou corrigí-los para facilitar a compreensão ou até mesmo adicionar testes não gerados automaticamente para cobrir uma situação não prevista no caso de uso.

Os testes manuais fornecem uma maior facilidade de leitura para o testador, devido à maior liberdade para a elaboração dos passos. Entretanto, com o intuito de facilitar a execução e tornar os testes mais independentes um do outro, muitos passos de testes se repetem em outros, gerando um grande potencial para a captura de registros de erros duplicados. Outra desvantagem é que os projetistas possuem diferentes estilos e pontos de vista em relação à criação dos casos de teste. A abordagem automática possibilita uma maior padronização dos testes, o que sem dúvida facilita o trabalho de execução.

Com a implementação completa do algoritmo proposto, os testes teriam uma maior facilidade de compreensão, seriam mais independentes entre si e teriam a mesma vantagem da padronização dos testes. No entanto, muitos passos também seriam repetidos, assim como na abordagem manual avaliada. Na realidade a forma como os testes são criados depende muito do estilo dos projetistas. É comum encontrar testes independentes uns dos outros, como encontrar testes que executam passos de outros testes ou que possuem outros casos de teste como pré-requisito. Enfim, essa questão se resume a uma decisão do projeto de testes.

Capítulo 7

Conclusões e Trabalhos Futuros

Atualmente o processo de testes é uma atividade estratégica e imprescindível para a garantia da qualidade de um projeto de software. O processo de testes de software é uma atividade que demanda muito esforço e, consequentemente, de alto custo. O objetivo geral deste trabalho foi reduzir o esforço do processo de testes.

Após um estudo das atividades mais comumente executadas em um processo de testes, foi identificada a oportunidade de reduzir o esforço da atividade de projeto de casos de testes através da aplicação de uma ferramenta de geração automática dos casos de testes baseada em modelos. Neste trabalho foi proposto um algoritmo de geração automática de casos de teste a partir de um documento de casos de uso. O algoritmo proposto gera casos de teste para os fluxos de casos de uso, ou seja, os fluxos principais, alternativos e de exceção. É necessário destacar que o algoritmo proposto se aplica apenas para testes caixa-preta e verificam apenas se o sistema se comporta de acordo com o especificado, não sendo capaz de gerar outros tipos de teste, como testes de estresse ou testes de carga, por exemplo.

Além de propor e implementar um algoritmo de geração automática de casos de teste, neste trabalho foi conduzido um experimento com o objetivo de avaliar os benefícios da utilização da abordagem de geração automática comparada à abordagem de elaboração manual. As contribuições deste trabalho, bem como as conclusões dos experimentos a respeito das vantagens e desvantagens da utilização da abordagem automática proposta para o projeto de testes, serão detalhadas na Seção 7.1.

O algoritmo foi implementado como um módulo de um sistema *web* de manutenção de casos de uso. Entretanto sua aplicação é muito mais abrangente, podendo este ser aplicado para diferentes tipos de aplicação e em diferentes plataformas. Não foi possível implementar todo o

algoritmo proposto devido algumas dificuldades não previstas encontradas no sistema *web* para o qual foi desenvolvido. Além das dificuldades de implementação, alguns problemas foram encontrados durante os experimentos. Mais informações a respeito dessas dificuldades serão relatadas na Seção 7.2.

Apesar de não ter sido possível implementar totalmente o algoritmo proposto, algumas vantagens foram observadas, e novas oportunidades foram descobertas durante os experimentos conduzidos em relação à implementação de parte do algoritmo. Na Seção 7.3 abordaremos com mais detalhes esse tópico, bem como outros possíveis trabalhos futuros.

7.1 Contribuições

Após a realização dos experimentos de avaliação, foi constatado que a abordagem automática obteve uma redução de esforço em relação à abordagem manual de 58%. Os experimentos realizados consideraram apenas os casos de uso de um requisito. Foi escolhido apenas um requisito para a elaboração dos projetos de teste devido às limitações de tempo. Entretanto, o resultado obtido é um forte indicativo de que a abordagem automática proposta representa uma grande redução no esforço total. A partir destes experimentos também é possível concluir que existe uma tendência de que quanto mais complexo o requisito e quanto mais fluxos associados, maior a redução do esforço da abordagem automática em relação à manual.

Outra importante contribuição dessa abordagem é a facilidade de manutenção dos casos de teste. Caso ocorra alguma atualização nos casos de uso, basta gerar os testes novamente para os casos de uso alterados e automaticamente todos os testes impactados pela atualização serão alterados.

As contribuições anteriores também proporcionam um aumento da produtividade da equipe de testes, pois os projetistas podem ser empregados em atividades mais intelectuais, ao invés de concentrar esforços na atividade de projeto de testes que muitas vezes é repetitiva e por esse motivo pode levar a introdução de erros. Com a geração automática é possível concentrar os esforços de testes na atividade de inspeção, para garantir uma maior qualidade dos testes e para elaborar casos de testes a partir da experiência dos projetistas que cobrem situações possíveis de caminhos no sistema que não estão devidamente especificadas no documento de casos de uso.

Uma outra contribuição da abordagem automática de casos de testes é uma maior padronização dos casos de testes gerados. Casos de testes elaborados manualmente são escritos dependendo do estilo do projetista e casos de teste para o mesmo requisito podem apresentar

formatos muito diferentes. Essas diferenças de estilo podem dificultar na atividade de execução de testes. Com a geração automática, os casos de testes serão gerados da mesma forma, criando assim uma padronização para os testes. Essa padronização auxilia na execução dos testes porque facilita a compreensão do testador. Entretanto, é importante destacar que apesar da padronização, a compreensão do caso de teste pode ser comprometida devido à forma como os casos de uso são escritos. Os casos de teste gerados pela abordagem automática refletem exatamente o que está escrito no caso de uso, ou seja, os passos tratam de ações do usuário na terceira pessoa. Os testes são mais facilmente compreendidos com frases imperativas. Essa dificuldade pode ser contornada se os testes forem alterados na etapa de inspeção.

A partir da execução dos experimentos foi constatado que a abordagem automática facilita muito na cobertura de todos os caminhos de execução possíveis de uma funcionalidade. A abordagem automática gerou testes para 11 caminhos que não foram verificados nos testes elaborados manualmente, mesmo após a etapa de inspeção. Mesmo com menos testes a abordagem automática verifica mais caminhos possíveis do que os testes da abordagem manual. Isso é possível porque os pré-requisitos dos testes indicam todos os testes que devem ser executados para se executar o teste em questão. Isso gera uma diminuição dos testes, por outro lado aumenta a dificuldade na atividade de execução.

7.2 Dificuldades Encontradas

Durante o desenvolvimento do projeto várias dificuldades foram enfrentadas. Dificuldades estas que não estavam prevista durante a definição do projeto e de seu escopo. O algoritmo de geração automática de casos de teste proposto, como detalhado na Seção 5.2 do Capítulo 5, é composto de três etapas. Entretanto, não foi possível implementar todas as etapas do algoritmo.

O algoritmo proposto foi implementado como um módulo do Sistema de Cadastro de Casos de Uso, detalhado no Capítulo 4. Quando o projeto foi definido, acreditávamos que o Sistema de Cadastro de Casos de Uso possuía algumas facilidades e necessidades para o tratamento dos dados necessários para geração dos casos de teste. Entretanto, durante o desenvolvimento, foi percebido que as associações entre os fluxos alternativos e de exceção e os passos do fluxo principal ao qual estavam associados não existiam. A estrutura da base de dados do sistema foi projetada com o intuito de facilitar o cadastro dos casos de uso. Os relacionamentos necessários para a geração automática de casos de teste de acordo com o algoritmo proposto por este trabalho não existiam.

O sistema associa os fluxos aos casos de uso corretamente. Os fluxos alternativos e de exceção podem surgir do fluxo principal do caso de uso, como também de outros fluxos. O sistema atualmente não é capaz de dizer qual é o fluxo origem de um fluxo alternativo ou de exceção. Fluxos alternativos e de exceção também podem retornar para algum passo de outro fluxo, que pode ser de qualquer tipo, seja principal, alternativo ou de exceção. Esse retorno também não é possível de ser registrado no sistema. Devido a esse problema, a implementação do algoritmo ficou comprometida. Seria necessária uma alteração em várias partes do sistema, desde a base de dados, para representar essas associações, a camada de negócio, para possibilitar o cadastro dos fluxos de origem e destino, até a interface, para permitir que o usuário cadastre essas informações. Essas profundas alterações no sistema não foram previstas inicialmente e não tinham como serem feitas com o tempo disponível.

Devido ao estado em que o sistema se encontrava foi possível apenas a implementação da primeira etapa do algoritmo que é a geração dos fluxos principal, alternativo e de exceção. Com essa alteração os casos de teste se tornaram limitados, pois eles dependem muito da execução de outros testes para serem devidamente executados, o que dificulta a execução dos testes e pode ocasionar um aumento do esforço desta atividade.

Apesar desta limitação, durante o experimento foi constatado que os testes gerados apenas com a primeira etapa do algoritmo possuíam algumas vantagens em relação aos testes que seriam gerados a partir da implementação total do algoritmo. Mais detalhes a respeito dessas vantagens podem ser encontrados na Seção 6.2.7 do Capítulo 6.

7.3 Trabalhos Futuros

Como proposta de trabalho futuro podemos citar a realização de experimentos com diferentes requisitos, desde requisitos com baixa complexidade a requisitos com alto grau de complexidade. Os experimentos conduzidos indicam que quanto maior a complexidade dos casos de uso, maior o ganho da abordagem automática proposta. Entretanto, é preciso comprovar essa tendência através da comparação das duas abordagens com requisitos mais complexos.

Outro experimento interessante seria elaborar projetos de testes utilizando as abordagens manual e automática para um sistema real e executar os testes projetados pelas duas abordagens. O esforço da execução das duas abordagens seriam medidos e com isso seria possível avaliar quais os impactos das duas abordagens no tempo de execução dos testes. Também será possível avaliar qual das duas abordagens tem maior potencial para capturar erros. Com esse experimento

poderíamos verificar se o ganho de esforço utilizando a abordagem automática não é perdido durante a execução dos testes. Poderíamos avaliar se além de reduzir o tempo do projeto de testes, a abordagem automática é capaz de capturar a mesma quantidade de erros ou, se possível, mais erros do que a abordagem manual.

Um outro importante trabalho a ser realizado seria a implementação das demais etapas do algoritmo proposto por este trabalho. Para tanto seria necessária uma modificação na estrutura da base de dados com o objetivo de possibilitar a implementação das demais etapas. Também seria necessário conduzir um experimento para avaliar as vantagens do algoritmo totalmente implementado em relação à abordagem manual.

Como descrito na seção anterior, apenas a implementação da primeira etapa do algoritmo foi possível. Apesar disso, durante os experimentos algumas vantagens foram percebidas após a análise dos testes gerados. Por não repetir passos presentes em outros testes, a geração automática utilizando a primeira etapa tem um menor potencial de gerar erros duplicados. Também se concluiu que mesmo com uma quantidade menor de casos de testes do que os testes gerados manualmente, os testes gerados automaticamente exercitam todos os caminhos possíveis de um requisito. Também seria relevante conduzir um experimento com as duas estratégias de abordagem automática para avaliar as vantagens e desvantagens de cada uma delas.

Também como trabalho futuro é possível citar a implementação de uma verificação na linguagem utilizada na geração dos testes. Atualmente os testes gerados apenas copiam os passos contidos nos casos de uso e os estruturam em formato de casos de teste. Com uma verificação da linguagem e modificação dos passos dos casos de uso para satisfazer as necessidades de testes, a qualidade dos testes aumentaria e o esforço necessário para inspeção dos testes gerados seria reduzido. O aumento na qualidade dos testes teria impacto direto na redução do esforço necessário para execução dos casos de teste.

Uma outra proposta seria desenvolver uma forma do sistema receber como entrada um documento de casos de uso escrito em um programa de edição de texto e gerar os casos de teste como saída. Ou até mesmo poderia ser desenvolvido um outro sistema que recebesse um documento de casos de uso em um programa de edição de texto como entrada. Com isso não seria necessário cadastrar o documento de casos de uso no Sistema de Cadastro de Casos de Uso reduzindo ainda mais o esforço do projeto de testes, visto que, de acordo com o experimento realizado, esta atividade equivale a 50% do esforço do projeto de testes utilizando a abordagem automática proposta.

Bibliografia

[1] MYERS, Glenford. J. **The Art of Software Testing.** 2. ed. Nova Jersey: John Wiley and Sons, 2004. 224 p.

[2] KANER, Cem; FALK, Jack; NGYEN, Hung Q. **Testing Computer Software.** 2. ed. Nova Jersey: John Wiley and Sons, 1999. 480 p.

[3] EL-FAR, I.K.; WHITTAKER, J. A. **Model-Based Software Testing. Encyclopedia of Software Engineering**. 2. ed. Nova Jersey: John Wiley and Sons, 2001. 1520 p.

[4] SINHA, A.;WILLIAMS, C. E.;SANTHANAM, P. A Measurement Framework for Evaluating Model-based Test Generation Tools. IBM Systems Jornal, v. 45, n. 3, p. 501-514. 2006.

[5] PRESSMAN, Roger S. **Engenharia de Software.** 6. ed. São Paulo: McGraw-Hill, 2006.

[6] INTHURN, Cândida. **Qualidade e Testes de Software.** Florianópolis: Visual Books, 2001.

[7] DUSTIN, Elfriede; RASHKA, Jeff; PAUL, John. **Automated Software Testing – Introduction, Management and Performance.** Nova York: Addison-Wesley, 1999.

[8] GELPERIN, D.; HETZEL, B. The Growth of Software Testing. Communications of the ACM, Volume 31 Issue 6, 1998.

[9] SWEBOK. **Guide to the Software Engineering Body of Knowledge.** Disponível em: <http://www.swebok.org> Acesso em: 15 de abril de 2008.

[10] LEWIS, William E. **Software Testing and Continuous Quality Improvement.** Auerbach, 2000.

[11] WILLIAMS, Laurie. **Testing Overview and Black-Box Testing Techniques.** 2004.

[12] BACH, James. What is Exploratory Testing? Disponível em: <http://www.satisfice.com/articles/what_is_et.shtml> Acesso em: 15 de abril de 2008.

[13] HALL, Marty; BROWN, Larry. **Core Servlets and JavaServer Pages.** 2. ed. Sun Microsystems Press/Prentice Hall PTR Book, 2003. 691 p.

[14] SILVA, Liana; SANTANA, Celio; ROCHA, Fernando; PASCHOALINO, Maíra; FALCONIERI, Gabriel; RIBEIRO, Lucio; WANDERLEY, Renata; SOARES, Sérgio; GUSMÃO, Cristine. Applying XP to an Agile Inexperienced Software Development Team. In: 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP2008), 2008, Limerick, Ireland.

[15] COCKBURN, Alistair. **Agile Software Development: The Cooperative Game.** 2 ed. Nova York: Addison-Wesley, 2006. 468 p.

[16] BECK, K. **Extreme Programming Explained – Embrace Change.** Nova York: Addison-Wesley, 2002.

[17] SCHNEIDER, Geri; WINTERS, Jason. **Applying Use Cases: A Practical Guide.** Nova York: Addison-Wesley, 1998.

[18] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software.** Nova York: Addison-Wesley, 1994. 416 p.

[19] Java 5.0. Disponível em: <http://java.sun.com/j2se/1.5.0/> Acesso em: 10 de maio de 2008.

[20] Eclipse – An Open Development Platform. Disponível em: <http://www.eclipse.org> Acesso em: 10 de maio de 2008.

[21] Apache Tomcat. Disponível em: <http://tomcat.apache.org> Acesso em: 10 de maio de 2008.

[22] MySQL: The World's Most Popular Open Source Database. Disponível em: <http://www.mysql.com> Acesso em: 10 de maio de 2008.

[23] JORGENSEN, Paul C. **Software Testing: A Craftman's Approach.** CRC, 1995.

[24] LARMAN, Craig. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.** 2 ed. Prentice Hall PTR, 2002. 627 p.

[25] AETG - Automatic Efficient Test Generator. Disponível em: <http://aetgweb.argreenhouse.com/> Acesso em: 10 de maio de 2008.

[26] MAURER, Peter. DGL - Data Generation Language. Disponível em: <http://cs.baylor.edu/~maurer/dgl.php> Acesso em: 10 de maio de 2008.

[27] HP WinRunner Software. Disponível em: <https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-127-24%5E1074_4000_100__> Acesso em: 10 de maio de 2008.

[28] Rational Robot. Disponível em: <http://www-306.ibm.com/software/awdtools/tester/robot/> Acesso em: 10 de maio de 2008.

[29] Telelogic Tau/Tester. Disponível em: <http://www.telelogic.com/products/tau/tester/index.cfm> Acesso em: 10 de maio de 2008.

[30] Test Complete – Automated Software Testing Tool. Disponível em: <http://automatedqa.com/products/testcomplete/> Acesso em: 10 de maio de 2008.

[31] TVEC – Test Vector Generation System. Disponível em: <http://www.t-vec.com/Home.asp> Acesso em: 10 de maio de 2008.

[32] Conformiq Qtronic General Purpose (GP). Disponível em: <http://www.conformiq.com/qtronic.php?variant=GP> Acesso em: 10 de maio de 2008.

[33] Model-Based Testing and Validation with Reactis. Disponível em: <http://www.reactive-systems.com/> Acesso em: 10 de maio de 2008.

[34] UniTESK. Disponível em: <http://www.unitesk.com/> Acesso em: 10 de maio de 2008.

[35] Simulink – Simulation and Model-Based Design. Disponível em: <http://www.mathworks.com/products/simulink/> Acesso em: 10 de maio de 2008.

[36] StateFlow – Design and simulate state machines and control logic. Disponível em: <http://www.mathworks.com/products/stateflow/> Acesso em: 10 de maio de 2008.

[37] ASML – Abstract State Machine Language Dsiponível em: <http://research.microsoft.com/fse/asml/> Acesso em: 10 de maio de 2008.

[38] GRIESKAMP, W.;GUREVICH, Y.; SCHULTE, W.; VEANES, M. Generating Finite State Machines from Abstract State Machines. Proceeding of ISSTA – International Symposium on Software Testing and Analysis. 2002.

Apêndice A

Caso de Uso 1 – Fazer Pedido

CDU001 – Fazer Pedido
Atores: Cliente
Pré-condição:
O usuário deve ter feito *log-in* e obtido autorização do sistema
Fluxo Principal:
1. O caso de uso começa quando o cliente seleciona "fazer pedido". 2. O sistema exibe a Tela de Fazer Pedido 3. O cliente fornece seu nome e endereço. 4. O cliente fornece os códigos do produto. 5. O sistema devolve as descrições e o preço de cada produto. 6. O sistema calculará os valores totais para cada produto fornecido. 7. O cliente fornece as informações sobre cartão de crédito. 8. O cliente submete os dados ao sistema. 9. O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento. 10. Quando o pagamento é confirmado, o pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente.
Pós-condição:
O pedido deve ter sido gravado no sistema e marcado como confirmado.

Fluxos Alternativos

FA001.01 – Fornecer CEP
1. No passo 3 do fluxo principal, o cliente fornece apenas o CEP. 2. O sistema coloca automaticamente a cidade e o estado. 3. O sistema retorna ao passo 4 do fluxo principal.

FA001.02 – Adicionar Item
1. No passo 4 do fluxo principal, o cliente pode selecionar “Adicionar Item”. 2. O sistema exibe o campo para adição do código de um produto. 3. O cliente fornece o código do produto. 4. O sistema fornece a descrição e o preço do produto. 5. O sistema atualiza o valor total. 6. O sistema retorna para o passo 4 do fluxo principal.

FA001.03 – Cancelar a Operação
1. A qualquer momento antes de submeter, o cliente pode selecionar cancelar. 2. O pedido não é gravado e o sistema retorna a tela principal

Fluxos de Exceção

FE001.01 – Informação incorreta
1. No passo 7 do fluxo principal, se alguma informação estiver correta, O sistema v pede ao cliente para corrigir a informação

Apêndice B

Caso de Testes Elaborados Manualmente Para o Caso de Uso 1

CT001 – Fazer Pedido com sucesso	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça nome e endereço	
3. Forneça os códigos do produto	O sistema devolve as descrições e o preço de cada produto e calcula os valores totais para cada produto fornecido
4. Forneça as informações sobre cartão de crédito	
5. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
6. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

CT002 – Fornecer CEP	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome	
3. Forneça apenas o CEP	O sistema coloca automaticamente a cidade e o estado
4. Forneça código do produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
5. Forneça as informações sobre o cartão de crédito	
6. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
7. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

CT003 – Adicionar Item	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome e endereço	
3. Forneça código do produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
4. Selecione "adicionar item"	
5. Forneça o código de outro produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
6. Forneça as informações sobre o cartão de crédito	
7. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
8. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

CT004 – Adicionar Itens	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome e endereço	
3. Forneça código do produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
4. Selecione "adicionar item"	
5. Forneça o código de outro produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
6. Selecione "adicionar item"	
7. Forneça o código de outro produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
8. Forneça as informações sobre o cartão de crédito	
9. Submeta os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento
10. Confirme o pagamento	O pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

CT005 – Dados de Cartão de Crédito incorretos	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome e endereço	
3. Forneça código do produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
4. Forneça informações incorretas sobre o cartão de crédito	
5. Submeta os dados ao sistema	O sistema verifica que os dados estão incorretos e pede ao cliente para corrigir a informação

CT006 – Cancelar a operação de Fazer Pedido	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Selecione "cancelar"	O pedido não é gravado e o sistema retorna a tela principal

CT007 – Cancelar a operação de Fazer Pedido após fornecer dados do cliente	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome e endereço	
3. Selecione "cancelar"	O pedido não é gravado e o sistema retorna a tela principal

CT008 – Cancelar a operação de Fazer Pedido após fornecer o código do produto	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome e endereço	
3. Forneça código do produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
4. Selecione "cancelar"	O pedido não é gravado e o sistema retorna a tela principal

CT009 – Cancelar a operação de Fazer Pedido após fornecer as informações do cartão de crédito	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome e endereço	
3. Forneça código do produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
4. Forneça as informações sobre o cartão de crédito	
5. Selecione "cancelar"	O pedido não é gravado e o sistema retorna a tela principal

CT010 – Dados de Produto incorreto	
Pré-requisitos:	
Efetuar *log-in* no sistema e obter autorização do sistema.	
Passos	Resultados Esperados
1. Selecione "fazer pedido"	O sistema exibe a Tela de Fazer Pedido
2. Forneça seu nome e endereço	
3. Forneça código do produto	O sistema fornece a descrição e preço do produto e atualiza o valor total
4. Forneça o código do produto inexistente	O sistema exibe a mensagem "Produto inexistente"

Apêndice C

Caso de Testes Gerados Automaticamente para o Caso de Uso 1

CT001 - Fazer Pedido	
Pré-requisitos: O usuário deve ter feito "log-in" e obtido autorização do sistema	
Passos	Resultados Esperados
1. O caso de uso começa quando o cliente seleciona "fazer pedido"	O sistema apresenta a tela de Fazer Pedido
2. O cliente fornece seu nome e endereço	
3. O cliente fornece código do produto	O sistema fornece a descrição e preço do produto. O sistema atualiza o valor total
4. O cliente fornece as informações sobre cartão de crédito	
5. O cliente submete os dados ao sistema	O sistema verifica as informações fornecidas, marca o pedido como "pendente" e envia as informações de pagamento para o sistema de contabilidade e pagamento. Quando o pagamento é confirmado, o pedido é marcado como "confirmado" e o número de pedido (NP) é dado ao cliente

CT002 - Fornecer CEP	
Pré-requisitos: O usuário deve ter feito "log-in" e obtido autorização do sistema. Execute o passo 1 do CT001	
Passos	Resultados Esperados
1. Se o cliente fornecer apenas o CEP	o sistema coloca automaticamente a cidade e o estado
2. Execute o CT001 a partir do passo 3	

CT003 - Adicionar Item	
Pré-requisitos: O usuário deve ter feito "log-in" e obtido autorização do sistema. Execute o CT001 até o passo 3	
Passos	Resultados Esperados
1. O cliente seleciona "adicionar item"	
2. O cliente fornece código do produto	O sistema fornece a descrição e preço do produto. O sistema atualiza o valor total
3. Execute o CT001 a partir do passo 4	

CT004 - Dados Incorretos	
Pré-requisitos: O usuário deve ter feito "log-in" e obtido autorização do sistema. Execute o CT001 até o passo 4	
Passos	Resultados Esperados
1. O cliente submete os dados incorretos ao sistema	O sistema pede ao cliente para corrigir a informação

CT005 - Cancelar pedido	
Pré-requisitos: O usuário deve ter feito "log-in" e obtido autorização do sistema. Execute esse caso de testes do passo 1 ao passo 4 do CT001	
Passos	Resultados Esperados
1. O cliente seleciona cancelar	O pedido não é gravado e o sistema retorna a tela principal