2 Estado da arte

Existem três conceitos importantes que serão abordados durante essa dissertação: geração automática de scripts teste a partir de casos de uso, desenvolvimento dirigido por comportamento e a técnica de "*capture and replay*". Foi realizada uma pesquisa na literatura para cada um desses tópicos. Na última seção será apresentada a diferença do que já foi realizado para o trabalho que está sendo desenvolvido.

2.1. Geração automática de scripts de teste a partir de casos de uso

Segundo (Cockburn *et al*. 2002), casos de uso são simplesmente histórias sobre como as pessoas usarão um sistema para realizar alguma tarefa e destaca 3 vantagens na utilização:

A primeira é que os casos de uso proporcionam um quadro semiformal para estruturar histórias e essa estruturação libera a criatividade das pessoas, tornando relativamente fácil para o usuário final de um sistema ler o documento com muito pouco treinamento.

A segunda vantagem é que os casos de uso descrevem os requisitos do sistema para as situações de erro. Uma vez que muito da complexidade do sistema encontra-se em lidar com situações de erro, descrever tais requisitos significa que as dificuldades associadas são detectadas e discutidas logo no início do ciclo de desenvolvimento.

Em terceiro lugar, embora que os casos de uso sejam essencialmente uma técnica de decomposição funcional, eles tornaram-se um elemento popular de desenvolvimento orientado a objetos de software. Várias pessoas, incluindo (Jacobson, 1992) e (Larman, 2002), descrevem metodologias para realizar os objetos necessários e implementar o comportamento descrito pelo caso de uso. Pode-se escrever um conjunto de casos de uso que descrevam o comportamento funcional do sistema e, em seguida, usar essas técnicas para projetar os objetos necessários para implementar esse comportamento.

Enfim, os casos de uso fornecem bons andaimes para pendurar outras informações de projeto. O gerente de projeto pode construir estimativas e cronogramas de liberação em torno deles. Designers de interface do usuário podem criar e vincular os seus desenhos nos casos de uso relevantes. Testadores podem construir cenários de teste das condições de sucesso e fracasso descrito nos casos de uso (Cockburn *et al*. 2002).

Sendo assim, é de suma importância que os casos de uso sejam verificados a fim de identificar possíveis defeitos inseridos no sistema. Uma maneira de reduzir o esforço e o custo na fase de teste de software, mas ainda assim preservando a sua eficácia, é a geração automática de casos de teste a partir de artefatos utilizados nas fases iniciais de desenvolvimento de software, tal como o caso de uso (Somé e Cheng 2008).

Em (Gutiérrez *et al.* 2007), os autores identificaram e classificaram as abordagens para gerar testes a partir de casos de uso. As abordagens podem ser divididas em três grupos, dependendo dos artefatos utilizados para a geração de casos de teste:

O primeiro grupo inclui abordagens que geram testes utilizando informações do formulário de caso de uso, como (Heumann, 2001). A abordagem enfatiza que a parte mais importante do caso de uso para gerar casos de teste é o fluxo dos eventos, em particular o fluxo básico de eventos e os fluxos alternativos de eventos. O fluxo básico de eventos deve cobrir o que "normalmente" acontece quando o caso de uso é realizado. Os fluxos alternativos de eventos cobrem o comportamento opcional ou excepcional caráter relativo ao comportamento normal, e também variações do comportamento normal, também podem ser entendidos como "desvios" do fluxo básico de eventos.

Na figura 1, (Heumann, 2001) apresenta a estrutura típica destes fluxos de eventos. A seta reta representa o fluxo básico de eventos, e as setas curvas representam os fluxos alternativos de eventos. Observe que alguns fluxos alternativos retornam ao fluxo básico de eventos, enquanto outros terminam o caso de uso. Tanto o fluxo básico de eventos quanto os fluxos alternativos de eventos devem ser estruturados em etapas ou sub-fluxos.

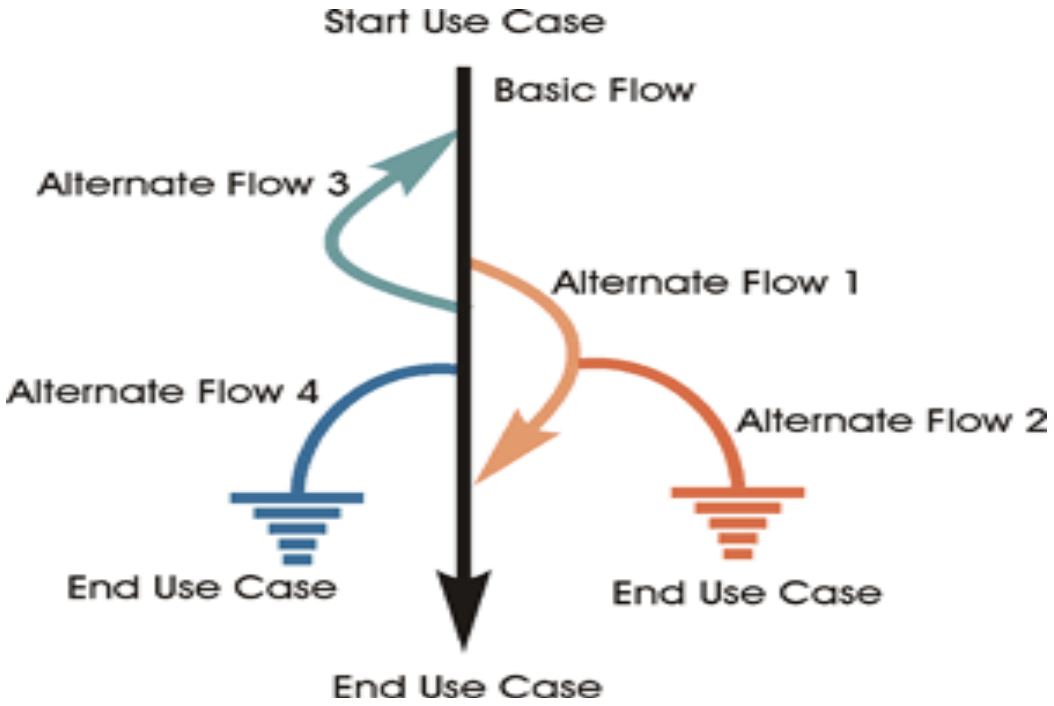

Figura 1 - Fluxos de Eventos - (Heumann, 2001).

Em sua abordagem, (Heumann, 2001) chama atenção para os métodos desordenados de concepção, organização e execução de atividades de teste, esses frequentemente levam a testar menos e consequentemente não atingem uma cobertura adequada. Ter um plano simples de como o teste é realizado, auxilia no aumento da cobertura, eficiência e consequentemente a medir a qualidade do software. Como ajuda, o autor descreve um processo de três passos para gerar casos de teste a partir do detalhamento de um caso de uso:

1. Para cada caso de uso, ler a descrição textual e identificar cada combinação de fluxos principais e alternativos - os cenários. Ou seja, gerar um conjunto de cenários dos casos de uso.

2. Para cada cenário identificado no passo anterior, identificar pelo menos um caso de teste, provavelmente haverá mais haverá mais de um.

3. Depois que todos os casos de teste foram identificados, estes devem ser analisados e validados para garantir a precisão e para identificar casos de teste redundantes ou em falta. Então, uma vez aprovados, o passo final é utilizar valores de dados reais como insumo para executar os testes e também verificar o resultado da obtido. Sem dados de teste, casos de teste (ou procedimentos de teste) não podem ser implementados ou executados, nem verificados, pois eles são apenas

descrições de condições e cenários. Portanto, é necessário identificar os valores reais a serem utilizados na execução dos testes e na verificação dos resultados.

Nessa abordagem, o autor ressalva que gerar casos de teste mais cedo no ciclo de desenvolvimento do software, permite a equipe identificar e reparar defeitos que seria muito caro reparar mais tarde, isso assegura que o sistema funcione de forma confiável e que ao utilizar uma metodologia claramente definida, os desenvolvedores podem simplificar o processo de testes, aumentar a eficiência e garantir a cobertura completa do teste.

O segundo grupo inclui abordagens que geram um "modelo de comportamento" a partir dos casos de uso e os testes deduzidos desses modelos, como (Labiche e Briand, 2002), (Nebut, 2006) ou (Ruder, 2004). Algumas das notações usadas neste grupo são: os diagramas de atividades, diagramas de máquinas de estado, sistema de transições de casos de uso ou sistemas ou árvores de cenários.

Em seu artigo, (Nebut, 2006) foca nos relacionamentos entre os casos de uso. O diagrama de caso de uso acrescido dos contratos (pré e pós condições) é utilizado com entrada para gerar um sequenciamento entre os casos de uso. Como nem todas as informações importantes para a geração estão descritas no diagrama de casos de uso, esse artigo também utiliza o diagrama de sequência. Após gerar o sequenciamento entre os casos de uso, é gerado um modelo de simulação através dos valores descritos nos dois diagramas. O resultado final e a geração de scripts de testes executáveis no formato *JUnit* (JUnit, 2011).

O terceiro grupo descreve abordagens centradas nas "variáveis e valores de teste" (Binder, 1999) e (Balcer e Ostrand, 1988), que consiste em identificar as variáveis operacionais que são explicitamente parte da interface que suporta o caso de uso, tais como, entradas e saídas do sistema, além de identificar os domínios das variáveis operacionais, ou seja, definir quais são os valores válidos e inválidos para cada variável. Em seguida desenvolver os relacionamentos operacionais, modelando o relacionamento entre as variáveis operacionais que determinam diferentes respostas do sistema. Podem-se modelar os relacionamentos através de uma tabela de decisão. E por fim desenvolver os casos de teste, escrevendo-os com base no relacionamento entre as variáveis operacionais. Os resultados esperados podem ser desenvolvidos pela observação

dos valores de entrada.

É consenso entre os trabalhos mencionados que o caso de uso é uma boa fonte para geração de casos de teste. Segue abaixo algumas das vantagens de utilizar casos de uso para gerar testes:

- A utilização dos casos de uso está consolidada nos processos de análise e projeto, facilitando o desenvolvimento dos casos de teste;
- Casos de uso refletem o ponto de vista do usuário;
- Provêm uma forma sistemática de desenvolvimento das informações necessárias para o projeto de teste;
- Casos de uso ambíguos, inconsistentes ou incompletos, são logo apontados pelos testes.

2.1.1. Limitações dos casos de uso

Alguns autores concordam que um caso de uso pode ser utilizado para derivar casos de testes (Fröhlich e Link 2000; Williams 2001; Dranidis, Tigka *et al*. 2003), e que casos de testes gerados a partir de casos de uso podem assegurar que os requisitos do sistema sejam atendidos (Dranidis, Tigka *et al*. 2003; Somé e Cheng 2008). Entretanto, o uso de casos de uso para geração de casos de testes tem as suas limitações, entre elas, estão: Casos de uso são escritos em uma linguagem natural informal, podendo gerar diferentes interpretações, propensas a erros e incompletude (Dranidis, Tigka *et al*. 2003; Somé e Cheng 2008).

Uma segunda limitação é garantir uma cobertura adequada para as sequências de ações de um caso de uso, ou seja, garantir com que os casos de testes gerados a partir das sequências de ações do caso de uso tenham um alto grau de probabilidade de encontrar defeitos no sistema (Somé e Cheng 2008).

Uma terceira dificuldade é que importantes restrições que definem a sequência entre um caso de uso e outro são expressas implicitamente, ou seja, elas são anotadas informalmente nos casos de uso (Somé e Cheng 2008), o que dificulta a automação e seleção de um conjunto adequado de casos de teste.

2.2. Desenvolvimento dirigido por comportamentos

O *BDD* - *Behaviour Driven Development* (Desenvolvimento Orientado-Dirigido por comportamento) é uma técnica que surgiu no contexto do desenvolvimento ágil e encoraja a colaboração entre desenvolvedores, testadores e participantes não técnicos do projeto de software. Criada por Dan North em 2003, ela permite especificar testes de aceitação a partir de informações contidas na descrição de uma estória.

Ele sugeriu uma linguagem semiestruturada composta por três palavras chaves: "Dado", "Quando" e "Então". Essas palavras devem ser empregadas de acordo com a seguinte estrutura: "dado um contexto, quando ocorrer determinado evento, então se deve obter determinado resultado". Essa estrutura formaria um cenário. Como exemplo, tem-se o seguinte cenário (e.g. o Administrador em Windows):

- Dado que estou autenticado como Administrador
- Quando eu clicar no botão Listar usuários
- Então será exibida a lista de todos os usuários cadastrados
- Quando eu selecionar nesta lista o usuário José e clicar no botão apagar
- Então será exibida a lista de usuários cadastrados e esta lista não conterá o usuário José

Associada a um cenário, também pode existir uma estória de usuário com as palavras "como", "eu quero" e "para" disposta no seguinte formato: "como um ator, eu quero determinada funcionalidade, para atingir determinado objetivo". Associada ao cenário exemplificado acima se pode ter a seguinte estória de usuário (e.g. o Administrador em Windows)::

- Como Administrador do sistema
- Eu quero ser capaz de adicionar, alterar e excluir os dados dos usuários
- Para poder manter o sistema atualizado

Dan North criou o primeiro *framework* do *BDD*, O *JBehave* - direcionado para linguagem *Java*, em seguida *RBehave* - para linguagem *Ruby*, que mais tarde foi integrado no projeto *RSpec*. O *RSpec* foi o primeiro *framework* baseado em estórias e depois substituído pelo *Cucumber*, desenvolvido principalmente por *Aslak Hellesøy*. Em 2008, *Chris Matts*, que esteve envolvido nas discussões em torno do desenvolvimento do primeiro *framework BDD*, surgiu a ideia de *Feature*

Injection, permitindo ao *BDD* cobrir o espaço de análise e fornecer um tratamento completo no ciclo de vida do software.

As práticas de *BDD* incluem:

- Envolver as partes interessadas no processo através de *Outside-In Development* (Desenvolvimento de Fora pra Dentro);
- Usar exemplos para descrever o comportamento de uma aplicação ou unidades de código;
- Automatizar os exemplos para prover um *feedback* rápido e testes de regressão;
- Usar "deve" (*should*) na hora de descrever o comportamento de software para ajudar esclarecer responsabilidades e permitir que funcionalidades do software sejam questionadas;
- Usar dublês de teste (*mocks, stubs, fakes, dummies, spies*) para auxiliar na colaboração entre módulos e códigos que ainda não foram escritos.

2.3.Geração de testes a partir de "*capture and replay*"

Ferramentas como *Squish for* Web (Squish, 2011) e *Selenium IDE* (Selenium, 2011) oferecem a capacidade de geração automática de testes para interfaces de sistemas web através de técnica conhecida como "*capture and replay*" (Araújo e Staa, 2009). Nesta técnica, as ações do usuário sobre a interface do sistema são gravadas por uma ferramenta e é usado um código que reproduz essas ações. Esse código então pode ser editado para refatoração e inserção de verificações, tais como assertivas. A figura 2 abaixo, foi gerada através da ferramenta *Selenium IDE* para ilustrar um exemplo da utilização da técnica "*capture and replay*" cujo cenário é realizar login em um sistema de webmail.

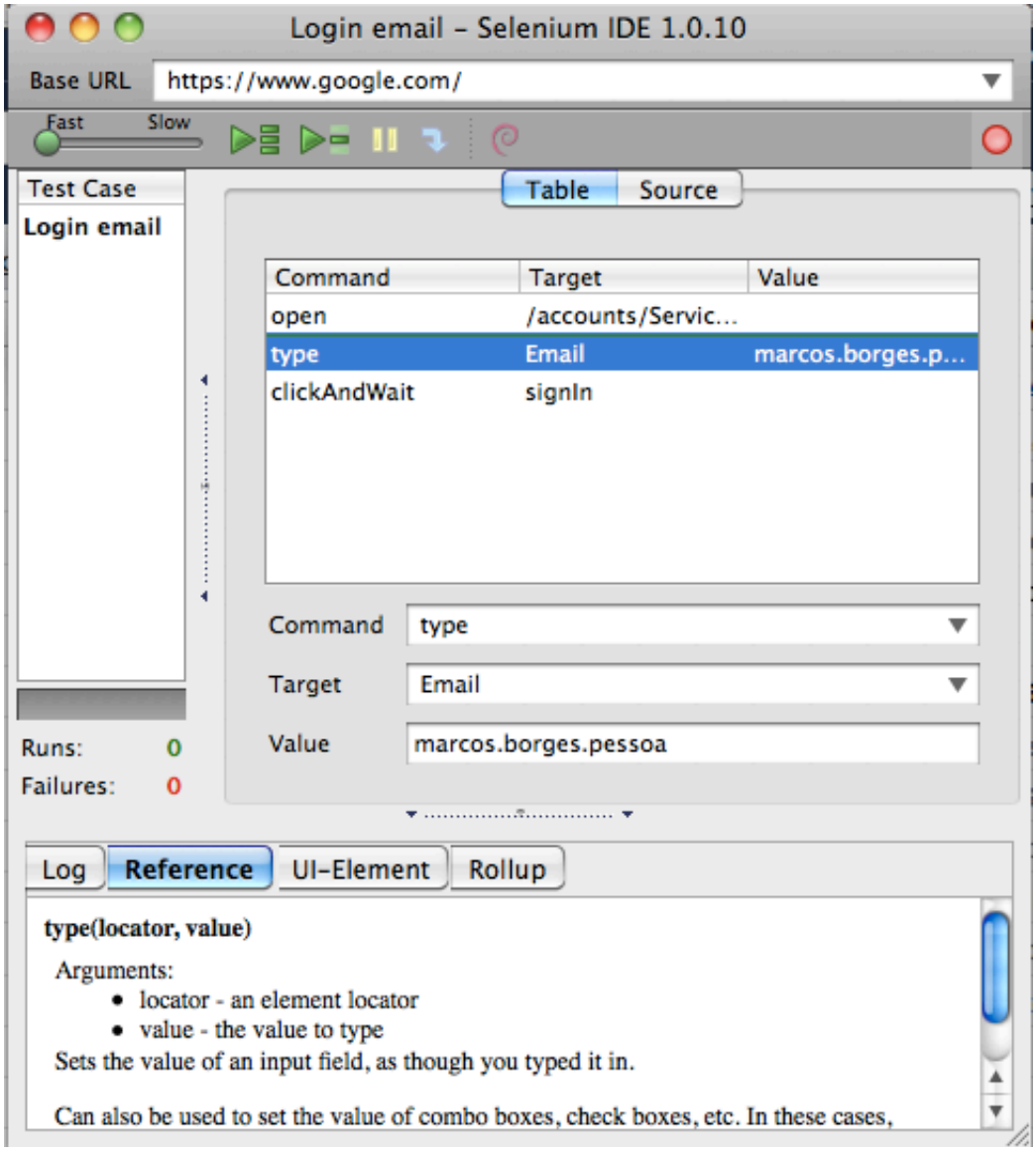

Figura 2 - Interface gráfica do *Selenium IDE* no *Mozilla Firefox* 3.6.

O grande ponto fraco da abordagem "*capture and replay*" é a sensibilidade a mudanças, tais como: mudanças no leiaute da tela, mudança dos campos do formulário e as mudanças nas regras de validade dos campos. As mudanças podem provocar a necessidade de repetir o *capture*. Além disso, realizar um *capture* é tedioso e tende a gerar capturas incorretas, obrigando a repetição. O que se quer então é gerar o script de teste, simulando o *capture*.

Outro ponto fraco é que o código gerado automaticamente acaba sendo mais difícil de compreender e manter do que o código escrito manualmente, caso não haja uma refatoração posterior à geração automática do código. Isso acontece porque, como todas as ações do usuário na interface são codificadas, acaba por existir código desnecessário, oriundo, por exemplo, de cliques de mouse

acidentais em elementos que não são de interesse para o teste, como uma área em branco da tela, ou ainda, codificação de ações que o usuário executou por engano. A figura 3 abaixo ilustra o código fonte exportado pela ferramenta *Selenium IDE* para o cenário capturado na figura 2. Neste caso a linguagem de programação escolhida para geração do código fonte foi *Ruby*.

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import java.util.regex.Pattern;

public class marcos extends SeleneseTestCase {
    @Before
    public void setUp() throws Exception {
        selenium = new DefaultSelenium("localhost", 4444, "*chrome",
        "https://www.google.com/");

        selenium.start();
    }

    @Test
    public void testMarcos() throws Exception {
        selenium.open("/accounts/ServiceLogin?service=mail&passive=true&rm=false
        &continue=http%3A%2F%2Fmail.google.com%2Fmail%2F%3Fui%3Dhtml%26zy%3Dl
        &bsv=llya694le36z&scc=1&ltmpl=default&ltmplcache=2");

        selenium.type("Email", "marcos.borges.pessoa");
        selenium.click("signIn");
        selenium.waitForPageToLoad("30000");
    }

    @After
    public void tearDown() throws Exception {
        selenium.stop();
    }
}
```

Figura 3 – Código *Ruby* gerado pelo *Selenium IDE.*

2.4. Diferenças da Dissertação

Por esse levantamento foi percebido que os casos de uso são uma boa fonte para geração de casos e cenários de teste e que cenários de teste podem trazer diversos benefícios na área de teste. Nas ferramentas e *frameworks* pesquisados foram identificadas dois tipos de abordagens. As que utilizam informações do caso de uso para gerar automaticamente casos de teste que são executados de forma manual. E as que os scripts são gerados de forma manual e executados automaticamente.

Em sua dissertação de mestrado, (Caldeira, 2010) propõe um processo e ferramentas para a geração semiautomática de scripts de teste funcional para

sistemas web, a partir de casos de uso e tabelas de decisão. Com o auxílio de uma ferramenta, monta-se manualmente uma tabela de decisão a partir desses casos de uso. Os casos de teste semânticos são gerados automaticamente a partir destas tabelas de decisão. Outra ferramenta é responsável por gerar os scripts de testes a partir dos casos de teste semânticos.

Existem muitas ferramentas e *frameworks* que propõem de alguma forma automatizar testes, é improvável que uma única ferramenta seja capaz de automatizar todas as atividades de teste. A maioria das ferramentas enfoca em uma determinada atividade ou grupo de atividades e algumas só abordam um aspecto de uma atividade. No entanto, não foi encontrado nenhum processo ou ferramenta que utilize informações do caso de uso para gerar e executar automaticamente scripts de teste e verificar se o comportamento da aplicação web está conforme descrito.