|  |
| --- |
| brasao |
| Marcos Borges Pessoa  Geração e execução automática de scripts de teste para aplicações web a partir de casos de uso direcionados por comportamento |
| Dissertação de mestrado  Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática do Departamento de Informática do Centro Técnico e Científico da PUC-Rio.  Orientador: Prof. Arndt von Staa |
|  |
| Rio de Janeiro, 30 de Agosto de 2011. |

|  |
| --- |
| brasao |
| Marcos Borges Pessoa  Geração e execução automática de scripts de teste para aplicações web a partir de casos de uso direcionados por comportamento |
|  |
| Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática do Departamento de Informática do Centro Técnico e Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada. |
| Prof. Arndt von Staa  Orientador e Presidente  Departamento de informática - PUC-Rio  Prof. Alessandro Fabricio Garcia  Departamento de informática - PUC-Rio  Prof. Gustavo Robichez de Carvalho  Departamento de informática - PUC-Rio  Prof.ª. Simone Diniz Junqueira Barbosa  Departamento de informática - PUC-Rio  Rio de Janeiro, 30 de Agosto de 2011. |

|  |  |
| --- | --- |
|  | Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador. |
|  | Marcos Borges Pessoa  Graduou-se em Bacharelado em Ciência da Computação pela Universidade de Alfenas (UNIFENAS) em Março de 2000. Tem experiência na área de Ciência da Computação, com ênfase em Desenvolvimento de Software. Tem trabalhado em analise de sistemas desde 2000. |
|  | Ficha Catalográfica |
|  | Marcos Borges Pessoa  Geração e execução automática de scripts de teste para aplicações web a partir de casos de uso direcionados por comportamento / Marcos Borges Pessoa; orientador: Arndt von Staa. - Rio de Janeiro: PUC-Rio, Departamento de Informática, 2011.  v., 100 f.: il. ; 29,7 cm  Dissertação de Mestrado - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática  Referencias bibliográficas incluídas.  Caso de uso; Teste de Software; Teste automatizado; Geração e Execução Automática de Scripts de Teste |

Aos meus pais e a minha esposa.

|  |
| --- |
| Agradecimentos |
| Aos meus pais, Tarlei Alves Pessoa e Maria Isabel Borges, pelo apoio, educação, carinho e dedicação.  A minha esposa Laura Dalia Farah Pessoa, pelo apoio, carinho e compreensão.  Ao meu orientador, prof. Arndt von Staa, por sua dedicação, ensinamentos e orientação.  À Globo.com, pelo financiamento e auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.  Aos professores da Comissão examinadora.  A todos os amigos e familiares que de alguma forma contribuíram para a realização deste trabalho. |

|  |
| --- |
| Resumo |
| Marcos Borges Pessoa. **Geração e execução de scripts de teste para aplicações web a partir de casos de uso.** Rio de Janeiro, 2011. 100p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.  Este trabalho visa explorar os requisitos de software, descritos na forma de casos de uso, como instrumento capaz de apoiar a geração e execução automática de testes funcionais, com o intuito de verificar automaticamente se o resultado obtido nos testes gerados e executados estão em conformidade com o especificado. O trabalho consiste em utilizar um processo e ferramenta para documentar casos de uso e automaticamente gerar e executar scripts de teste para verificar o comportamento funcional de aplicações web. As informações do caso de uso, em especial os fluxos de eventos (principal e alternativos), devem ser estruturados obedecendo um “modelo de comportamento” para que seja possível armazenar os dados e utilizá-los como entrada na integração com a ferramenta de testes. Neste trabalho, utilizou-se a ferramenta *Selenium* para a automação da interação com o navegador. A avaliação da eficácia deu-se através da aplicação do processo em sistemas reais e através de comparação com outras técnicas aplicadas nos mesmos sistemas. |
| Palavras-chave  Caso de uso; Teste de Software; Teste automatizado; Geração e Execução Automática de Scripts de Teste. |

|  |
| --- |
| Abstract |
| Marcos Borges Pessoa. **Automatic generation and execution of test scripts for web applications from use case driven by behavior.** Rio de Janeiro, 2010. 100p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.  This work aims at exploring the software requirements, described in the form of use cases, as an instrument to support the automatic generation and execution of functional tests, in order to automatically check if the result obtained in the tests generated and executed are in accordance with specified. The work is to use a process and tool for documenting use cases and automatically generates and executes test scripts to verify the behavior of web applications. The information of the use case, especially the flow of events (main and alternative), should be structured obeying a "behavior model" so that can store the data and use them as input for integration with the testing tool. In this work, we used the tool Selenium to automate the interaction with the browser. The assessment of efficacy was made through the application process in real systems and by comparison with other techniques applied in the same systems. |
| Keywords  Use Case, Software Testing, Automated Testing, Automated Generation and Execution of Test Scripts |

|  |
| --- |
| Sumário |
|  |

1 Introdução 14

1.1. Objetivo da Dissertação 17

1.2. Geração e execução automática dos testes 17

1.3. Organização da Dissertação 18

2 Estado da arte 19

2.1. Geração automática de scripts de teste a partir de casos de uso 19

2.2. Desenvolvimento dirigido por comportamentos 24

2.3. Geração de testes a partir de “*capture and replay*” 25

2.4. Diferenças da Dissertação 27

3 Processo 29

3.1. Redação dos casos de uso 29

3.2. Armazenamento do caso de uso 36

3.3. Geração e execução automática de testes 36

4 Desenvolvimento da ferramenta 38

4.1. Requisitos funcionais 38

4.2. Requisitos não funcionais 43

4.3. Diagrama de casos de uso 44

4.4. Modelo de dados 45

4.5. Persistência do dados 46

4.6. Gerador e executor de scripts de teste 51

4.7. Resultado da execução dos testes 60

4.8. Diagrama de implantação 62

4.9. Decisões de projeto 62

5 Estudo de caso 65

5.1. Visão geral das aplicações utilizadas nos testes 65

5.2. Escopo do experimento 68

5.3. Situação atual 69

5.4. Resultados obtidos 70

6 Conclusões e trabalhos futuros 81

6.1. Trabalhos futuros 83

7 Bibliografia 84

|  |
| --- |
| Lista de imagens |
|  |

Figura 1 - Fluxos de Eventos, básico e alternativos - (Heumann, 2001). 21

Figura 2 - Interface gráfica do *Selenium IDE* no *Mozilla Firefox* 3.6*.* 26

Figura 3 – Código *Ruby* gerado automaticamente pelo *Selenium IDE.* 27

Figura 4 – Visão geral do processo de redação, geração e execução dos testes*.* 29

Figura 5 – Processo de redação do caso de uso*.* 30

Figura 6 – Padrão para redação das ações do usuário. 31

Figura 7 – Padrão para redação das verificações*.* 32

Figura 8 – Formulário para redação do caso de uso – 1 de 4*.* 33

Figura 9 – Formulário para redação do caso de uso – 2 de 4*.* 34

Figura 10 – Formulário para redação do caso de uso – 3 de 4*.* 35

Figura 11 – Formulário para redação do caso de uso – 4 de 4*.* 35

Figura 12 – Armazenamento das informações do caso de uso*.* 36

Figura 13 – Geração e execução automática dos testes*.* 37

Figura 14 – Diagrama dos casos de uso da ferramenta*.* 44

Figura 15 – Modelo de dados da ferramenta. 45

Figura 16 – Fluxo para cadastrar um caso de uso*.* 46

Figura 17 – Formulário para cadastrar um projeto na ferramenta*.* 47

Figura 18 – Formulário para redação do caso de uso – 1 de 2. 47

Figura 19 – Formulário para redação do caso de uso – 1 de 2. 48

Figura 20 – Passo do tipo comentário. 49

Figura 21 – Passo do tipo “ação”. 49

Figura 22 – Diferentes opções do passo “ação”. 49

Figura 23 – Passo do tipo “verificação”. 50

Figura 24 – Projeto armazenado na tabela “Projects”. 50

Figura 25 – Caso de uso armazenados na tabela “Usecase”. 50

Figura 26 – Fluxos armazenados na tabela “Fluxos” 50

Figura 27 – Passos armazenados na tabela “Passos” . 50

Figura 28 – Fluxo para gerar e executar automaticamente os testes. 52

Figura 29 – Registros da tabela “Passos” usados para testar os fluxos de autenticação na “Central de Relacionamentos”. 52

Figura 30 – Passo do tipo “ação” para acessar a vitrine de produtos. 53

Figura 31 – Tela da vitrine de produtos e serviços do e-commerce. 54

Figura 32 – Passo do tipo “ação” para selecionar o produto na vitrine. 54

Figura 33 – Tela de detalhe do produto selecionado na vitrine. 55

Figura 34 – Passo do tipo “ação” para adicionar o produto no carrinho. 55

Figura 35 – Tela do carrinho de compras. 55

Figura 36 – Passos do tipo “ação” utilizados no carrinho de compras. 55

Figura 37 – Tela para informar os dados de identificação. 56

Figura 38 – Passos do tipo “ação” utilizados na tela de identificação. 56

Figura 39 – Tela para informar os dados de cadastro – 1 de 2. 57

Figura 40 – Passos do tipo “ação” utilizados no tela de cadastro 1 de 2. 57

Figura 41 – Tela para informar os dados de cadastro – 2 de 2. 58

Figura 42 – Passos do tipo “ação” utilizados no tela de cadastro 1 de 2. 59

Figura 43 – Tela de sucesso na realização da compra. 59

Figura 44 – Passo do tipo “verificação” utilizado na tela de sucesso. 59

Figura 45 – Visão do resultado da geração e execução dos testes de todos os projetos. 60

Figura 46 – Visão do resultado da geração e execução dos testes para os casos de uso do projeto. 61

Figura 47 – Visão do resultado da geração e execução dos testes para os fluxos e passos do caso de uso. 61

Figura 48 – Diagrama de implantação da ferramenta. 62

Figura 49 – Visão geral das aplicações utilizadas no experimento. 67

Figura 50 – Visão testes criados manualmente através do Eclipse IDE (Eclipse, 2011). 76

Figura 51 – Código *JUnit* para invocar a execução dos testes criados manualmente. 77

Figura 52 – Exemplo de um caso de teste criado manualmente utilizando Java com *Selenium* *API*. 77

Figura 53 – Fluxo básico de eventos do caso de uso UC1. 80

|  |
| --- |
| Lista de tabelas |
| Tabela 1 – Leiaute do formulário de caso de uso proposto por (Staa, 2010), adaptado de (Cockburn, 2000). 31  Tabela 2 – Padrões para tipos de elementos e suas ações. 32  Tabela 3 – Padrões para tipos de elementos e suas verificações. 32  Tabela 4 – Esforço para redigir os casos de uso. 71  Tabela 5 – Estimativa do esforço para redigir os casos de teste identificados. 75  Tabela 6 – Esforço para criar os casos de teste manualmente. 78  Tabela 7 – Esforço criar os casos de teste através da técnica “*capture and replay*”. 79 |

# Introdução

Uma parcela significativa do tempo e do custo utilizados na construção dos produtos de software é consumida pelos testes. Com a crescente complexidade e tamanho do software e a necessidade de entregas rápidas, a expectativa é que a quantidade de testes necessários vá continuar a aumentar. Além disso, são necessárias grandes melhorias na tecnologia de testes e o teste de software automatizado é uma das respostas mais promissoras (Dustin *et al.* 2009).

Para se desenvolver um software podemos fazer diferentes tipos de teste, com diversos objetivos - performance, carga, funcionalidade, estresse, entre outros (Karner *et al.* 1999).

Os testes funcionais são testes derivados da especificação do software ou componente, ou aqueles que o testador se preocupa somente com a funcionalidade do sistema e não com sua implementação, são os mais utilizados devido à necessidade que os softwares produzidos façam o que foi acordado com o cliente na fase de análise de requisitos (Sommerville, 2003).

Já a execução de testes de software pode ser feita tanto de forma manual quanto automatizada. A execução manual consiste na reprodução por uma pessoa do teste previamente definido e documentado. Já a execução automática consiste na automação do processo de teste manual atualmente em uso (Zambelich, 2006). Scripts de teste devem ser construídos para reproduzir os testes que serão executados automaticamente.

Os testes funcionais devem ser automatizados quando são muito repetitivos e demandam um esforço considerável de tempo quando realizados manualmente (Zambelich, 2006). A realização de testes automatizados, além de possibilitar a redução do ciclo de testes, permite um aumento indireto da cobertura do software e, consequentemente, da sua qualidade, porque permite que os testadores foquem seus esforços em outros tipos de teste ou em testes que não possam ser automatizados.

Algumas das características dos processos de desenvolvimento de software que seguem o paradigma de metodologias ágeis, como o “*extreme programming*” (Beck e Andres, 2004), são a entrega frequente de novas funcionalidades, ciclos de desenvolvimento curtos e rapidez de resposta a mudanças nos requisitos ao longo do projeto e foco em manter o software funcionado ao invés de documentação abrangente. Uma das consequências do desenvolvimento iterativo com ciclos curtos e da tolerância a mudanças é a necessidade de testes frequentes, em particular, torna-se necessária uma elevada frequência de execução de testes de regressão, a fim de determinar se novas funcionalidades não comprometeram o correto comportamento de outros requisitos anteriormente implementados e já validados.

Uma das grandes diferenças entre projetos tradicionais e de desenvolvimento ágil é que os processos tradicionais geralmente são bastante prescritivos e temos que documentar tudo que estiver definido no processo, o que geralmente é muita coisa. Em métodos ágeis não há prescrição de documentação, mas isso não significa que não há necessidade de documentar desde que seja necessário de fato. Muitas pessoas confundem isso e dizem que nunca se deve documentar em projetos ágeis, o que é um grande engano. Em projetos ágeis podemos documentar, desde que seja necessário de fato. A ideia é não perder tempo com nada que não seja requerido de verdade para o projeto.

Para (Highsmith, 2009), a atividade de documentar tem que ser rápida, não pode dar trabalho. Usar ferramentas como wikis, geradores de documentação podem ajudar. Se for fácil documentar, as chances de fazê-lo serão maiores. Se a documentação for fácil de ser acessada (e tiver busca) ela será mais útil. Além disso, prefira usar uma tecnologia fácil e conhecida para que todos os membros do time possam documentar. Por exemplo, se você escolher usar *Latex*, podemos reduzir as chances de designers documentarem.

Como integrante da mesma equipe de desenvolvimento de software há cinco anos, e que nos últimos três anos migrou da metodologia de desenvolvimento tradicional para uma metodologia de desenvolvimento ágil, observamos no âmbito do nosso ambiente de desenvolvimento ágil, alguns problemas relacionados a testes e documentação, tais como:

* documentação escassa e muitas vezes inexistente;
* documentos sem padronização;
* documentos descentralizados;
* documentos difíceis de encontrar;
* dificuldades técnicas para automatizar os testes funcionais;
* necessidade de executar manualmente um grande número de testes funcionais a cada nova release do software;
* testes funcionais executados de forma desordenada, uma vez que existem poucos documentos de requisitos para se basear. Um ponto forte é que a cada execução novos caminhos são testados, um ponto fraco é a dificuldade em realizar os testes de regressão;
* pouca ou nenhuma contribuição dos envolvidos não técnicos na geração e execução de testes funcionais, como é o caso do cliente;
* dentro outros.

Vimos então, a necessidade e a oportunidade de estudos visando desenvolver maneiras mais eficazes para facilitar e incentivar a documentação de requisitos, bem como meios de minimizar o esforço e as dificuldades da equipe de desenvolvimento para automatizar testes funcionais. Pensamos em utilizar documentos de caso de uso como instrumento capaz de apoiar a equipe na documentação dos requisitos funcionais e na geração e execução automática de testes funcionais, com o intuito de verificar automaticamente se o resultado obtido nos testes gerados e executados está em conformidade com o especificado.

O trabalho consiste em pesquisar na literatura uma forma de utilizar uma abordagem de casos de uso direcionados por comportamento para documentar requisitos de software e a partir dessas informações automaticamente gerar e executar scripts de teste para verificar o comportamento funcional de aplicações web. As informações do caso de uso, em especial os fluxos de eventos, devem ser estruturados obedecendo a um “modelo de comportamento” para que seja possível armazenar os dados e utilizá-los como entrada na integração com o *framework* de testes. Levando em conta a disponibilidade e a experiência já adquirida, foi escolhida a ferramenta *Selenium* (Selenium, 2011), que será responsável por executar automaticamente os scripts de teste com a interface web.

Ao documentar um requisito utilizando a abordagem apoiada pela ferramenta o fluxo principal e cada fluxo alternativo do caso de uso será um cenário a ser coberto por testes automáticos.

## Objetivo da Dissertação

Essa pesquisa tem como objetivo especificar, projetar, desenvolver uma ferramenta para viabilizar a documentação de requisitos funcionais na forma de casos de uso, que possa ser utilizada mesmo por pessoas com pouco ou nenhum treinamento em computação, bem como utilizar dados dessa documentação para gerar e executar automaticamente scripts de testes e verificar se o comportamento da aplicações web estão em conformidade que o que foi descrito e avaliar os seguintes itens dentro de um cenário de desenvolvimento real:

* esforço para redigir os casos de uso;
* esforço para assegurar que tenham sido criados suficientes scripts de teste;
* comparar com testes executados manualmente;
* comparar com testes gerados e executados por outra ferramenta;
* relatar as dificuldades encontradas.

A expectativa é:

* minimizar o problema da falta de documentação;
* proporcionar confiança e dar mais agilidade, uma vez que os principais testes funcionais serão executados automaticamente e darão retorno rápido para que a equipe possa decidir sobre a transferência da release para o ambiente de produção;
* melhorar a coerência entre a documentação e o funcionamento do software, uma vez que o caso de uso será capaz de verificar o comportamento nele descrito.

## Geração e execução automática dos testes

Como parte do processo, o cadastro dos casos de uso pode ser realizado por qualquer membro da equipe do projeto, com ou sem conhecimento técnico, e os dados serão armazenados pela ferramenta em um sistema gerenciador de banco de dados. Após realizar o cadastro do caso de uso, a ferramenta pode utilizar os passos informados no fluxo básico e nos fluxos alternativos para montar scripts de teste e integrar ao *framework* de testes *Selenium* (Selenium, 2011) para executar a interação com a interface web.

A ferramenta armazena o resultado de cada script executado e durante a execução é possível acompanhar o resultado dos testes e verificar rapidamente se o comportamento da aplicação está em conformidade com o que foi descrito na forma de caso de uso.

## Organização da Dissertação

O restante deste documento está organizado em cinco capítulos, da seguinte forma:

O **Capítulo 2** apresenta o estado da arte para geração de testes automáticos a partir de casos de uso e as ferramentas e *frameworks* existentes comparados a ferramenta proposta.

O **Capítulo 3** descreve uma visão macro do processo de documentação dos casos de uso e da geração e execução automática dos testes.

O **Capítulo 4** descreve a ferramenta desenvolvida para apoiar na aplicação da processo de geração automática dos testes, os requisitos funcionais e não funcionais necessários e desejados, diagramas, modelo de dados, persistência dos dados e como os testes são gerados, executados e o resultado pode ser visualizado.

O **Capítulo 5** apresenta os resultados da utilização da abordagem proposta em um cenário real.

O **Capítulo 6** apresenta as conclusões e trabalhos futuros.

# 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.

## 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 afim 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.

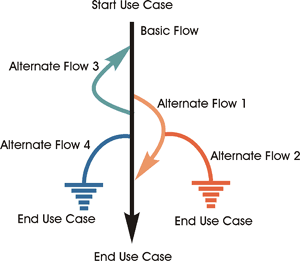


Figura 1 - Fluxos de Eventos, básico e alternativos - (Heumann, 2001).

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 do 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.

Top of Form

1. Depois que todos os casos de teste foram identificados, devem ser analisados ​​e validados para garantir a precisão e para identificar casos de teste redundantes ou em falta. Então, uma vez que sejam 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, 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. Pode-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;
* Provêem 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.

### 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.

## 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 com 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.

## 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.

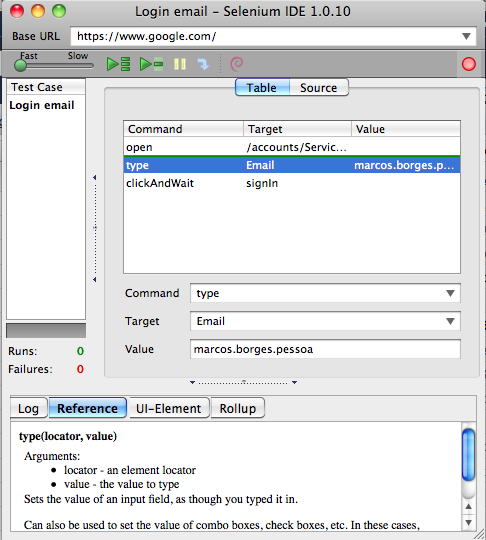


Figura 2 - Interface gráfica do *Selenium IDE* no *Mozilla Firefox* 3.6*.*

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 foi *Ruby*.



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

## 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.

# Processo

Neste capítulo será apresentado e justificado o processo de documentação e de testes que foi desenvolvido para auxiliar o desenvolvimento ágil a gerar documentos de requisitos e diminuir o esforço e as barreiras tecnológicas para criar testes funcionais para aplicações web. Como pode ser visualizado na figura 4 abaixo, o processo inicia quando o usuário cadastra o documento de caso de uso. Ao realizar o cadastro do caso de uso, os dados informados no cadastro são armazenados em um sistema gerenciador de banco de dados.

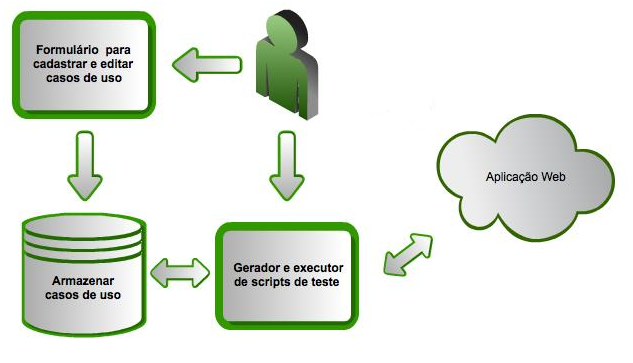


Figura 4 – Visão geral do processo de redação, geração e execução dos testes*.*

Após armazenar os dados do caso de uso no banco de dados, o usuário pode solicitar a geração e execução dos testes. Os testes serão gerados e executarão uma interação com a aplicação web para validar se o comportamento está conforme descrito nos casos de uso. Durante a geração e execução dos testes, se algum teste falhar, o usuário será informado e apresentado ao erro.

## Redação dos casos de uso

Dentre as várias informações que um caso de uso pode apresentar, está a descrição dos fluxos de eventos principais e secundários descrevendo a interação dos atores com o sistema. É comum o uso de linguagem natural na descrição destes fluxos, pois tal redação facilita o entendimento da funcionalidade por pessoas que não possuem treinamento em computação, como pode ser o caso do cliente. Este formato também permite que o cliente acompanhe a formulação do caso de uso juntamente com o analista e valide a sua descrição. No entanto, a linguagem natural pode dar margem a interpretações errôneas e a ambiguidades na descrição do caso de uso.

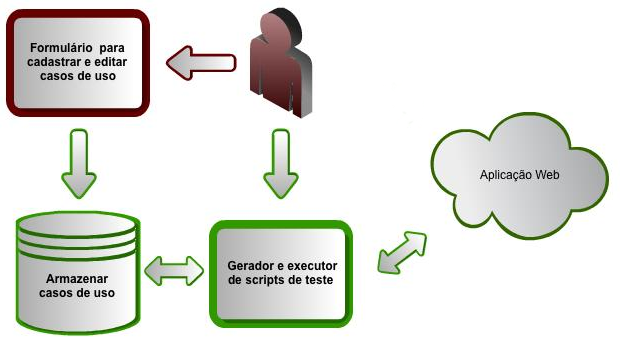


Figura 5 – Processo de redação do caso de uso*.*

Para evitar esse problema, a linguagem utilizada deve ser um português estruturado (Díaz et al, 2004) (Staa, 2010), a fim de evitar a ocorrência de ambiguidades, mas que seja de fácil entendimento para quem não tem treinamento em computação. A tabela 1 abaixo apresenta o leiaute de formulário, proposto por (Staa, 2010) e adaptado de (Cockburn, 2000), que foi utilizado para documentar os casos de uso na abordagem proposta nesta dissertação. Dentre os campos apresentados na tabela, os de maior interesse para auxiliar na geração e execução dos testes são:

* Fluxo principal: que apresenta quais os passos esperados na execução normal do caso de uso;
* Fluxos alternativos: que fornecem os passos executados em comportamentos alternativos ou em caso de algum erro, desvio ou exceção relativos ao fluxo principal.

O restante dos campos são utilizados somente para compor a documentação.

|  |  |  |
| --- | --- | --- |
| Caso de uso | Nome (identificação) do caso de uso. | |
| Resumo | Descrição resumida do objetivo principal. | |
| Escopo | O que é abrangido pelo caso de uso. | |
| Atores | Nome (identificação) do ator | Descrição do interesse ou objetivo do ator |
| Invariante | Condições que deverão estar satisfeitas antes e após a execução do caso de uso. | |
| Pré-condições | Condições que precisam estar satisfeitas antes de iniciar o caso de uso. | |
| Fluxo principal | Sequência de ações esperadas. | |
| Fluxos alternativos | Sequências de ações que tratam de exceções e desvios do fluxo principal. | |
| Pós-condições | Condições que devem estar satisfeitas ao terminar a execução normal do caso de uso. | |
| Garantia mínima | Condições que devem estar satisfeitas sempre, independente de como termine o caso de uso. | |
| Requisitos | Requisitos adicionais, tais como requisitos não funcionais e outras características desejáveis. | |
| Casos de uso correlatos | Relação de casos de uso correlacionados com o presente caso de uso. | |

Tabela 1 – Leiaute do formulário de caso de uso proposto por (Staa, 2010), adaptado de (Cockburn, 2000).

Cada frase da descrição dos passos do fluxo principal ou dos fluxos secundários deve ser de um dos 3 tipos:

* **Ação**: cada interação que o usuário pode executar com o sistema;
* **Verificação**: resposta do sistema às ações do usuário;
* **Documentação**: utilizada apenas para complementar o texto do fluxo de eventos.

Assim, como proposto em (Díaz *et al*, 2004), a descrição do fluxo principal e secundários deve ser através de períodos simples e com substantivos, ou expressões substantivas, que designem uma única entidade.

Dado que queremos automatizar a interação do usuário com o sistema, o padrão de redação de cada passo do fluxo principal e do fluxo secundário deve seguir um formato que permita identificar qual elemento da interface será exercitado pelo usuário. Para facilitar esta identificação o padrão de escrita adotado para “ações” do usuário na interface deve ser o da figura 6 abaixo.



Figura 6 – Padrão para redação das ações do usuário.

E o padrão de “verificação” das respostas do sistema às ações do usuário deve seguir o exemplo da figura 7 abaixo. Para ambos padrões de escrita dos passos, ação e verificação, os espaços “<...>” não são obrigatórios e são livres para entrada de qualquer texto, afim de facilitar a escrita do passo. O tipo de passo “documentação” não segue padrões e permite a entrada de qualquer texto.



Figura 7 – Padrão para redação das verificações*.*

Neste trabalho estamos interessados em casos de uso para sistemas com interface web, assim os elementos de interface e seus respectivos valores para “Padrão de ação” são:

|  |  |
| --- | --- |
| Tipo de Elemento | Padrão de ação |
| Campo de entrada de texto. | “preenche” |
| Elementos que disparam ação através de cliques, como botões e link. | “clica” |
| *Checkbox* e *Radio Buttons* | “marca a opção” |
| Seletores | “seleciona” |

Tabela 2 – Padrões para tipos de elementos e suas ações.

Os elementos de interface e seus respectivos valores para “Padrão de verificação” são:

|  |  |
| --- | --- |
| Tipo de elemento | Padrão de verificação |
| Página Web | “abre a página“ |
| Mensagem | “apresenta a mensagem” |
| Texto | “apresenta o texto” |

Tabela 3 – Padrões para tipos de elementos e suas verificações.

Tomemos como exemplo o cadastro de um caso de uso cujo requisito é realizar a compra de um produto através do e-commerce de um provedor da internet. Apesar dos vários fluxos possíveis, no exemplo foram utilizados apenas dois fluxos: o fluxo principal e um fluxo alternativo. O fluxo principal descreve os passos para realizar a compra do produto através do e-commerce utilizando cartão de crédito como forma de pagamento e o fluxo alternativo descreve os passos para realizar a compra do mesmo produto utilizando débito automático como forma de pagamento. A figura 8, a figura 9, a figura 10 e a figura 11 juntas formam o formulário completo do caso de uso cadastrado.



Figura 8 – Formulário para redação do caso de uso – 1 de 4*.*



Figura 9 – Formulário para redação do caso de uso – 2 de 4*.*



Figura 10 – Formulário para redação do caso de uso – 3 de 4*.*

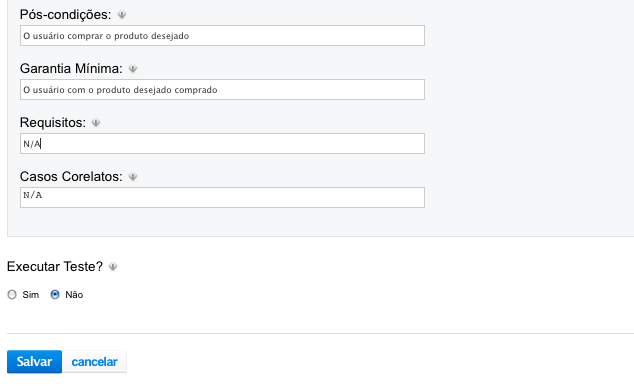


Figura 11 – Formulário para redação do caso de uso – 4 de 4*.*

## Armazenamento do caso de uso

Após preencher os dados do caso de uso, conforme descrito no item 3.1, o usuário solicita que os dados sejam armazenados em um sistema gerenciador de banco de dados.

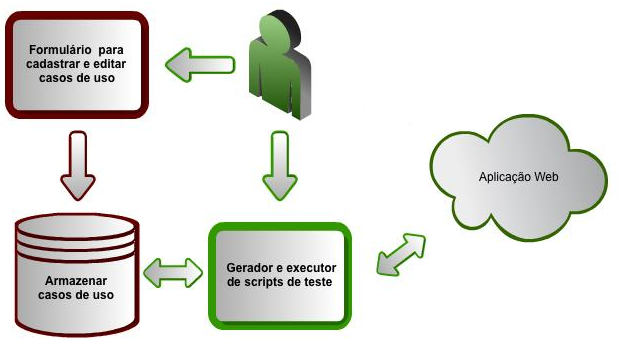


Figura 12 – Armazenamento das informações do caso de uso*.*

A ferramenta foi desenvolvida em *Ruby on Rails* que utiliza o padrão de arquitetura *ActiveRecord* (Flower, 2003), para armazenar os dados em um banco de dados relacional. Após o armazenamento das informações do caso de uso em tabelas do banco de dados, este pode então ser editado para possíveis correções ou adições de fluxos e passos. Para melhor entendimento de como os dados são persistidos pela ferramenta, ver no capítulo 4 os itens 4.4 e 4.5.

## Geração e execução automática de testes

Nesta etapa, o usuário solicita que a ferramenta inicie a geração e execução dos testes com a aplicação web. A ferramenta inicia o processamento e busca no banco de dados as informações do caso de uso, em especial os passos do fluxo básico e dos fluxos alternativos, para montar os scripts de testes e integrar com o módulo do *Selenium* (Selenium, 2011) para executar o teste na interface web.

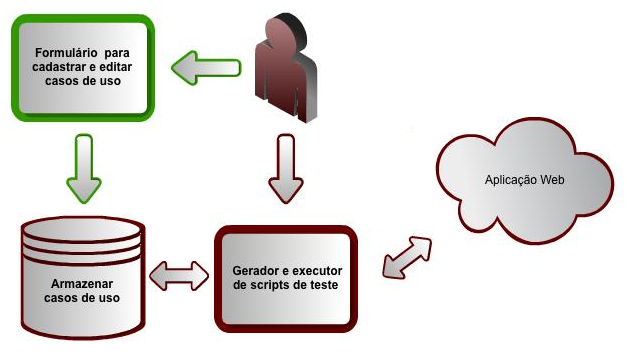


Figura 13 – Geração e execução automática dos testes*.*

Para cada passo do tipo “ação” e “verificação”, descritos no item 3.1, é gerado um script que executa uma interação com a aplicação web através da integração com o módulo do *Selenium*. Durante o processo de geração e execução dos testes, cada passo gerado e executado pode ser visualizado através da janela do navegador utilizado pelo *Selenium* e após a execução o resultado de cada passo executado é atualizado no banco de dados. Após gerar e executar os scripts correspondentes a cada passo dos fluxos de eventos do caso de uso de um determinado projeto, o usuário pode visualizar o resultado da execução dos testes nos formulários disponíveis na ferramenta. Para melhor entender o processo de geração e execução dos testes, ver detalhamento no capítulo 4, no item 4.6.

# Desenvolvimento da ferramenta

Visando facilitar a tarefa de documentar requisitos funcionais e de gerar testes automáticos em uma única ferramenta para proporcionar mais agilidade ao desenvolvimento de software, pesquisamos e verificamos na literatura que o caso de uso é uma boa forma para documentar as funcionalidades do software, é fácil de escrever e de ser entendido por pessoas com pouco conhecimento técnico, além de ser uma boa fonte para gerar casos de teste.

Pensamos então em especificar, projetar, desenvolver uma ferramenta para facilitar a documentação dos casos de uso, utilizando uma abordagem direcionada por comportamentos e um português restrito para descrever os fluxos de eventos (principal e secundários) para que seja possível integrar com um *framework* de testes e executar automaticamente testes funcionais em aplicações web e verificar se o comportamento está conforme descrito.

Neste capítulo serão apresentados os requisitos e a arquitetura necessários para que a ferramenta possa ajudar a minimizar o problema da falta de documentação e a dificuldade em automatizar testes, ambos observados e já discutidos anteriormente em capítulos anteriores neste documento.

## Requisitos funcionais

O processo pensado para documentar casos de uso e gerar e executar testes automaticamente com informações dessa documentação através da ferramenta, consiste em realizar três funcionalidades principais: O cadastro das informações do caso de uso, a geração e execução dos scripts de testes e a visualização do resultado. Abaixo a descrição dos requisitos desejados para que a ferramenta possa auxiliar na execução dessas atividades.

### Cadastrar projetos

* O usuário informa que deseja cadastrar um projeto;
* A ferramenta deve exibir o formulário para cadastrar o projeto;
* O usuário preenche o formulário com os dados para criar o projeto e confirma o cadastro;
* A ferramenta deve cadastrar os dados do projeto, retornar para lista de projetos, conforme o requisito de “Visualizar projetos” e exibir uma mensagem de sucesso.

### Visualizar projetos

* O usuário informa que deseja visualizar os projetos cadastrados;
* O ferramenta deve exibir uma lista dos projetos cadastrados e para cada projeto as seguintes informações: nome, descrição, quantidade de casos, data de criação, data de alteração e se o projeto esta ativo ou inativo para gerar e executar automaticamente os scripts de teste.

### Filtrar projetos

* Na lista de projetos descrita no requisito acima, “Visualizar projetos”,o usuário informa o nome ou parte do nome do projeto e solicita a pesquisa;
* A ferramenta deve pesquisar e apresentar uma lista dos projetos encontrados no mesmo formato da lista descrita no requisito funcional “Visualizar projetos”.

### Alterar projetos

* O usuário informa o projeto que deseja alterar;
* A ferramenta deve apresentar o formulário com os dados do projeto;
* O usuário altera os dados do projeto e confirma a alteração;
* A ferramenta deve salvar as alterações realizada nos dados no projeto, retornar para lista de projetos, conforme descrito no requisito funcional “Visualizar projetos”, e exibir uma mensagem de sucesso.

### Cadastrar casos de uso

* O usuário informa que deseja cadastrar um caso de uso para um determinado projeto;
* A ferramenta deve exibir o formulário para cadastrar o caso de uso;
* O usuário informa os dados e confirma o cadastro;
* A ferramenta deve cadastrar o caso de uso para o projeto informado, retornar para lista de casos de uso, conforme o requisito de “Visualizar casos de uso” e exibir uma mensagem de sucesso.

### Visualizar casos de uso

* O usuário informa que deseja visualizar os casos de uso cadastrados para um determinado projeto;
* O ferramenta deve exibir uma lista dos casos de uso cadastrados para o projeto e para cada caso de uso apresentar as seguintes informações: nome, quantidade de fluxos de eventos, quantidade de fluxos que geraram e executaram corretamente os scripts de teste, quantidade de fluxos que geraram e executaram incorretamente os scripts de teste, data de alteração e se o caso de uso esta ativo ou inativo para gerar e executar automaticamente os scripts de teste.

### Filtrar casos de uso

* Na lista de casos de uso descrita no requisito acima, “Visualizar casos de uso”,o usuário informa o projeto e o nome ou parte do nome do caso de uso e solicita a pesquisa;
* A ferramenta deve pesquisar e apresentar uma lista dos casos de uso encontrados para o projeto informado e nome informados no mesmo formato da lista descrita no requisito funcional “Visualizar casos de uso”.

### Alterar casos de uso

* O usuário informa o caso de uso do projeto que deseja alterar;
* A ferramenta deve apresentar o formulário com os dados do caso de uso;
* O usuário altera os dados do caso de uso e confirma a alteração;
* A ferramenta deve salvar as alterações nos dados no caso de uso para o projeto selecionado, retornar para lista de casos de uso, conforme descrito no requisito funcional “Visualizar casos de uso”, e exibir uma mensagem de sucesso.

### Gerar e executar automaticamente scripts de teste para um determinado projeto

* O usuário solicita a geração e execução automática dos scripts de teste de um determinado projeto.
* A ferramenta deve iniciar o servidor (*Selenium-Server*) e realizar os seguintes operações para cada caso de uso cadastrado no projeto com status “ativo”:
  + buscar o fluxo básico e o(s) fluxo(s) s alternativo(s);
  + Para cada fluxo, buscar os passos;
  + Para cada passo, gerar um script de teste, integrar ao framework de testes (*Selenium-Client*) e executar a interação com a interface web;
  + Armazenar o resultado da execução de cada script;
  + Finalizar o servidor (*Selenium-Server*).

.

### Gerar e executar automaticamente scripts de teste para todos os projetos

* O usuário solicita a geração e execução automática dos scripts de teste para todos os casos de uso de todos os projetos cadastrados.
* A ferramenta deve iniciar o servidor (*Selenium-Server*) e realizar os seguintes operações para cada projeto cadastrado com status “ativo:
  + buscar os casos de uso também com status “ativo” para o processo de geração e execução automática de scripts de teste;
  + Para cada caso de uso, buscar o fluxo básico e os(s) fluxo(s) s alternativo(s);
  + Para cada fluxo, buscar os passos;
  + Para cada passo, gerar um script de teste, integrar ao framework de testes (*Selenium-Client*) e executar a interação com a interface web;
  + Armazenar o resultado da execução de cada script;
  + Finalizar o servidor (*Selenium-Server*).

### Visualizar o resultado da geração e execução dos testes

* O usuário solicita visualizar o resultado da geração e execução dos scripts de teste executados através de um dos requisitos funcionais “Gerar e executar automaticamente scripts de teste para um determinado projeto” ou “Gerar e executar automaticamente scripts de teste para todos os projetos” ;
* A ferramenta deve apresentar o resultado da geração e execução dos scripts de teste através de 3 visões, afim de facilitar a identificação de possíveis falhas ocorridas durante o processo:
  + **Visão do geral** – listar todos os projetos cadastrados e as seguintes informações para cada projeto: nome, quantidade de casos de uso, quantidade de fluxos de eventos, quantidade de fluxos que geraram e executaram corretamente os scripts de teste, quantidade de fluxos que geraram e executaram incorretamente os scripts de teste e data da última execução.
  + **Visão do projeto** – listar os casos de uso de um projeto, conforme descrito no requisito funcional “Visualizar casos de uso”.
  + **Visão do caso de uso** – mesma visão descrita no requisito funcional “alterar caso de uso” com a situação da execução de cada passo.

## Requisitos não funcionais

Abaixo a lista dos requisitos não funcionais pensados para o desenvolvimento da ferramenta:

* A ferramenta deve ser uma aplicação web para facilitar a distribuição e ser disponibilizada na intranet da empresa durante a utilização do experimento.
* Deve ter uma interface simples e de fácil uso para permitir agilidade no cadastro da documentação.
* Deve suportar o acesso simultâneo de pelo menos três usuários dos seis usuários que vão interagir com a ferramenta, dentre eles: quatro desenvolvedores, um coordenador e um cliente. Ambos fazem parte de uma equipe de desenvolvimento ágil e têm conhecimento avançado de internet;
* Ser compatível com os seguintes sistemas operacionais utilizados na empesa:
  + *Linux* - distribuição *UBUNTU* – versão 9 ou superior;
  + *Windows*  - versão VISTA ou superior;
  + *Mac OS X* – versão 10 ou superior;
* Ser compatível com os seguintes navegadores e versões utilizados na empresa:
  + *Mozilla Firefox* – versão 3 ou superior;
  + *Safari* – versão 4 ou superior;
  + *Internet Explorer* – Versão 7 ou superior;
* Deve utilizar apenas tecnologias de softwares livre, tais como:
  + Banco de dados;
  + Linguagens e *frameworks* de programação;
  + Servidores de aplicação;
  + Ferramentas de desenvolvimento;
  + Gerenciadores de versão;
  + Ferramentas de *build* e *deploy*;

## Diagrama de casos de uso

A figura 14 abaixo ilustra a interação do usuário com os casos de uso e relação entre os casos de uso. Como todos os atores interessados podem interagir com todos os casos de uso do sistema, esses atores foram generalizados em um ator “Usuário”.



Figura 14 – Diagrama dos casos de uso da ferramenta*.*

Na relação entre os casos de uso, temos:

* “Filtrar projetos” utiliza o caso de caso de uso “Visualizar projetos” para apresentar o resultado do filtro realizado através do nome do projeto.
* “Filtrar casos de uso” utiliza o caso de uso “Visualizar casos de uso” para apresentar o resultado do filtro realizado através do nome do caso de uso.
* “Gerar e executar testes para todos os projetos” utiliza o caso de uso “Gerar e executar testes por projeto” para realizar a gerar e executar os testes em todos os projetos cadastrados.
* “Visualizar resultado da geração e execução dos testes” utiliza o caso de uso “Visualizar casos de uso“ para apresentar o resultado dos testes por casos de uso do projeto e também utiliza o caso de uso “Alterar caso de uso” para apresentar o resultado de cada passo dos fluxos do caso de uso.

## Modelo de dados

A figura 15 abaixo apresenta o modelo de dados desenhado para suportar a ferramenta no armazenamento das informações do caso de uso e na extração das informações para montar os scripts de teste.

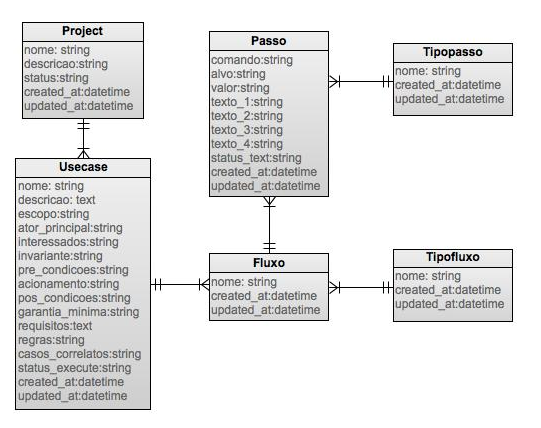


Figura 15 – Modelo de dados da ferramenta.

## Persistência do dados

Conforme mencionado anteriormente, no capítulo 3, no item 3.2, a ferramenta utiliza a arquitetura *ActiveRecord* (Flower, 2003), para armazenar os dados em um banco de dados relacional. Nesta arquitetura, uma tabela do banco de dados é representada por uma classe e uma instância desse objeto está vinculada a uma única linha na tabela. Após a criação de um objeto, uma nova linha é adicionada à tabela e qualquer objeto carregado obtém suas informações do banco de dados. Quando um objeto é atualizado a linha correspondente na tabela também é atualizada.

O fluxograma da figura 16 abaixo ilustra os passos para cadastrar um caso de uso através da ferramenta. O primeiro passo é cadastrar o projeto no qual o caso de uso será inserido, isso porque a organização dos casos de uso é feita através dos projetos.

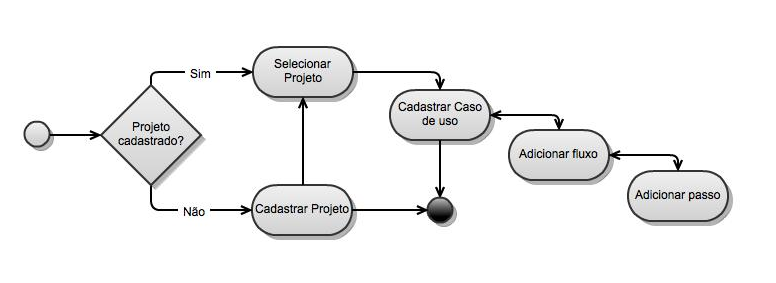


Figura 16 – Fluxo para cadastrar um caso de uso*.*

O cadastro do projeto deve ser realizado pela ferramenta através da funcionalidade “Cadastrar projetos”, conforme descrita nos requisitos funcionais. Ao finalizar o cadastro, os dados são armazenados na tabela “Project” apresentada no modelo de dados da figura 15. Vale ressaltar que o atributo “status” do projeto, pode ter apenas os dois valores: “S” ou “N“. Esse atributo diz respeito ao status do projeto no processo de execução dos testes. Caso tenha o valor “S”, o projeto será utilizado na geração e execução de testes. Caso contrário, não será utilizado no processo. A figura 17 abaixo apresenta um exemplo do formulário para cadastrar projeto de um projeto já cadastrado.

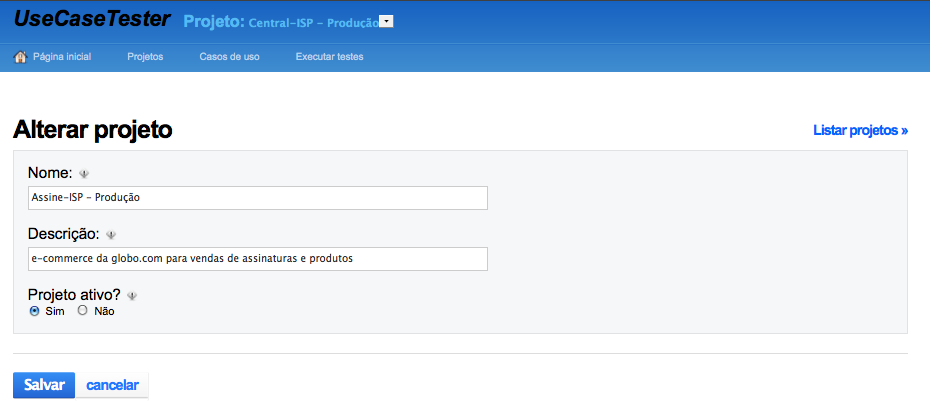


Figura 17 – Formulário para cadastrar um projeto na ferramenta*.*

Após cadastrar o projeto pode ser iniciado o cadastro dos casos de uso. O cadastro do caso de uso deve ser realizado através da funcionalidade “Cadastrar caso de uso”, descrita como requerimento funcional neste capítulo. O formulário para cadastrar caso de uso é responsável por manipular os dados em três tabelas: “Usecase”, “Fluxos” e “Passos”, ou seja, ao cadastrar um caso de uso, os dados são distribuídos nessas três tabelas. A figura 18 e a figura 19 abaixo apresentam um exemplo do formulário de um caso de uso já cadastrado para o projeto “Central-ISP – Produção”, selecionado no menu superior do formulário, e como os dados foram distribuídos nas tabelas do modelo.

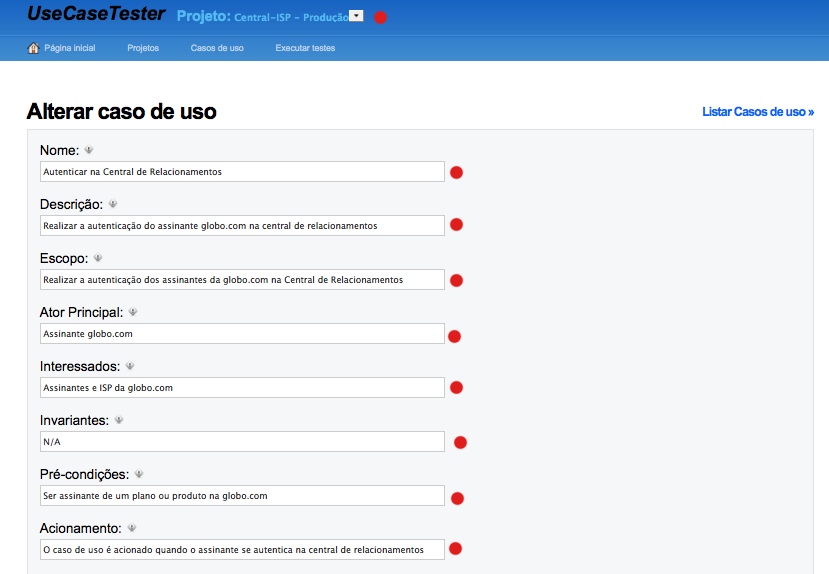


Figura 18 – Formulário para redação do caso de uso – 1 de 2.

As informações assinaladas com um círculo na cor vermelho estão armazenados na tabela “Usecase”, as assinaladas com a cor amarelo estão armazenadas na tabela de “Fluxos” e as assinaladas na cor preto estão armazenadas na tabela de “Passos”. Assim como o projeto, o caso de uso tem uma propriedade “status\_execute” que pode ter apenas dois valores: “S” ou “N“. Esse atributo diz respeito ao status do caso de uso para execução dos testes. Caso tenha o valor “S”, o caso de uso será utilizado na geração e execução de testes. Caso contrário, não será utilizado no processo.

Os fluxos do caso de uso podem ser de dois tipos: Fluxo principal e Fluxo alternativo. Um caso de uso deve ter um fluxo principal e pode ter quantos fluxos alternativos forem necessários. Os fluxos de eventos podem receber o cadastro de três tipos de passos para descrever seu comportamento, são eles: “comentário”, “ação” e “verificação”.

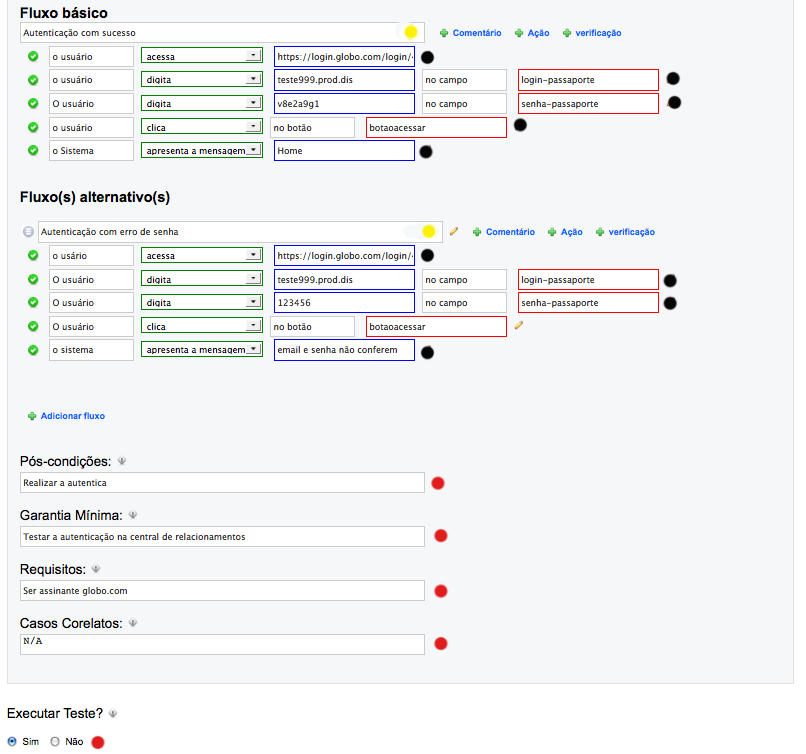


Figura 19 – Formulário para redação do caso de uso – 1 de 2.

**Comentário** – o passo comentário, conforme ilustrado abaixo na figura 20 é um campo texto livre para digitação e pode ser utilizado para enriquecer a documentação do fluxo. Passos desse tipo não tem utilidade no processo de geração dos testes, apenas complementam a documentação.



Figura 20 – Passo do tipo comentário.

**Ação** – cada passo do tipo ação cadastrado no fluxo de eventos será responsável por realizar uma interação com o formulário web através da integração com o *framework* de testes *Selenium*. A figura 21 abaixo ilustra a inclusão desse tipo de passo dentro do fluxo e também exibe a lista de opções com os tipos de ações disponíveis.

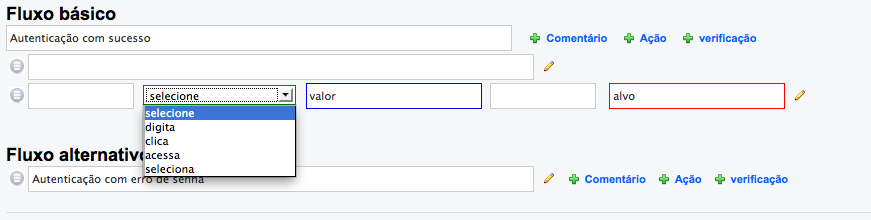


Figura 21 – Passo do tipo “ação”.

A configuração dos campos do passo do tipo ação pode ser alterada dependendo do tipo de ação escolhida para o passo na lista de seleção. O exemplo da figura 22 abaixo apresenta a configuração do passo para cada tipo de ação disponível na lista de seleção apresentada na figura 21.

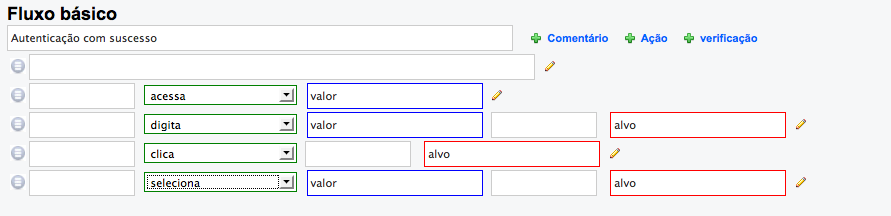


Figura 22 – Diferentes opções do passo “ação”.

**Verificação** – esse tipo de passo é responsável por verificar, através da integração com o *framework* de testes *Selenium*, se o texto cadastrado no passo existe no formulário web da aplicação. Isso possibilita que a ferramenta verifique as mensagens apresentadas pela aplicação web e consiga verificar se o fluxo atingiu o objetivo esperado. Abaixo na figura 23 um exemplo da inclusão desse tipo de passo em um fluxo.



Figura 23 – Passo do tipo “verificação”.

Para os três tipos de passos apresentados (documentação, ação e verificação), assim como na manutenção de fluxos, os passos incluídos no fluxo podem ser excluídos e também reordenados dentro do próprio fluxo através do recurso *drag-and-drop* (arrastar e largar) implementado no formulário.

A figura 24, a figura 25, a figura 26 e a figura 27 abaixo ilustram os dados armazenados nas tabela do banco de dados após realizar o fluxo completo de cadastro de um caso de uso. Na figura 24 estão as informações do projeto, na figura 25 as informações do caso de uso, na figura 26 dos fluxos e na figura 27 as informações dos passos.



Figura 24 – Projeto armazenado na tabela “Projects”.



Figura 25 – Caso de uso armazenados na tabela “Usecase”.



Figura 26 – Fluxos armazenados na tabela “Fluxos”

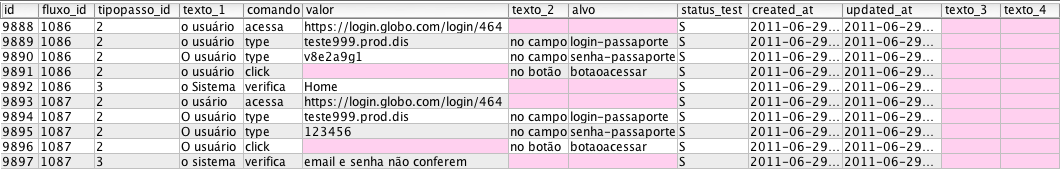


Figura 27 – Passos armazenados na tabela “Passos” .

## Gerador e executor de scripts de teste

Após realizar o cadastro dos casos de uso a ferramenta pode gerar os scripts de teste e integrar ao *framework* de testes *Selenium* para executar a interação com a interface web e assim verificar o comportamento da aplicação web. O fluxograma da figura 28 abaixo é um retrato do código implementado para gerar e executar os scripts de testes. As atividades em branco, são as atividades que a ferramenta faz a integração com o *framework* *Selenium* para manipular o *Selenium-Server* e o *Selenium-Client*.

Quando inicia a geração e execução dos testes a primeira atividade executada pela ferramenta é iniciar *Selenium-Server*, essa atividade inicializa a parte servidora do *framework* para receber as futuras requisições *HTTP’s* provindas dos scripts que serão gerados pela ferramenta e executados através dos *Selenium-Client*. O *Selenium-Server* é mantido ativo até que todos os testes sejam executados, após terminar, o *Selenium-Server* é encerrado.

Após inicializar o *Selenium-Server* a ferramenta verifica se a solicitação para gerar e executar os testes foi feita para um projeto ou para todos os projetos. Quando a solicitação é feita para executar testes em todos os projetos a ferramenta faz a busca no banco de dados pelos projetos cadastrados com atributo “status=S” e então realiza o mesmo fluxo para cada projeto.

Em seguida a ferramenta busca pelos casos de uso do projeto com atributo “status\_execute=S”. Para cada caso de uso é iniciada uma instância do navegador através do *Selenium-Client* e a instância é mantida até que todos os fluxos do caso de uso sejam executados.

O próximo passo é buscar os fluxos do caso de uso e para cada fluxo criar uma nova sessão do navegador que foi iniciado no passo anterior. A sessão é mantida até que todos os passos do fluxo sejam executados. Criar uma nova sessão do navegador para cada fluxo garante que não haja interferência de dados armazenados na sessão do navegador pelo fluxo anterior.

Por fim, a ferramenta busca os passos do fluxo e para cada passo do fluxo monta um script de teste e integra ao *selenium-client* para que o mesmo possa executar uma interação com a interface web.

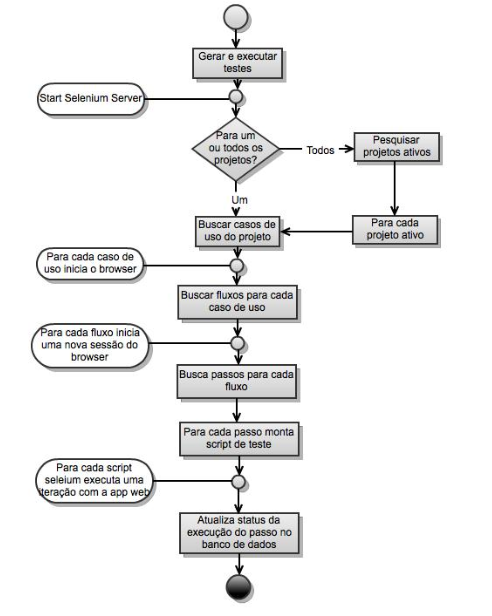


Figura 28 – Fluxo para gerar e executar automaticamente os testes.

Abaixo um exemplo de como a ferramenta utiliza as informações da tabela “passos”, ilustrados na figura 29 para montar os scripts no padrão do *Selenium-Client*.

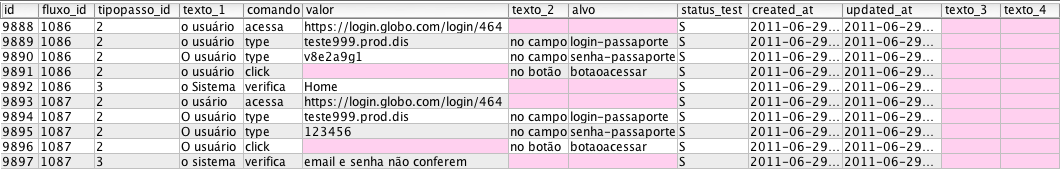


Figura 29 – Registros da tabela “Passos” usados para testar os fluxos de autenticação na “Central de Relacionamentos”.

Scripts gerados para o fluxo – Autenticar com sucesso:

* *Selenium.open(“https://login.globo.com/login/464”);*
* *Selenium.type(“login-passaporte”, “teste999.prod.dis”);*
* *Selenium.type(“senha-passaporte”, “123456”);*
* *Selenium.click(“botaoacessar”);*
* *Selenium.isTextPresent?(“Home”);*

Scripts gerados para o fluxo – Autenticar com erro de senha:

* *Selenium.open(“https://login.globo.com/login/464”);*
* *Selenium.type(“login-passaporte”, “teste999.prod.dis”);*
* *Selenium.type(“senha-passaporte”, “654321”);*
* *Selenium.click(“botaoacessar”);*
* *Selenium.isTextPresent?(“email ou senha não conferem”);*

Ao executar cada passo a ferramenta recebe um retorno do *Selenium-Client* indicando se o *script* gerado foi executado com sucesso ou não, caso o script tenha sido executado com sucesso a ferramenta atualiza a propriedade “status\_test” do passo no banco de dados para “S”, caso contrário, para “N”. É através deste propriedade do passo que a ferramenta apresenta o resultado dos testes.

As figuras abaixo, entre a figura 30 e a figura 44 apresentam a sequência das telas e dos passos utilizados em cada tela para realizar execução da compra de um produto no e-commerce de um provedor de internet. A execução inicia com passo ilustrado na figura 30, utilizado para acessar a vitrine de produtos do e-commerce.



Figura 30 – Passo do tipo “ação” para acessar a vitrine de produtos.

Script gerado e utilizado na integração com o *Selenium*:

1. *Selenium.open*(“http://assine.globo.com”);

As figura 31 abaixo, apresenta a vitrine de produtos carregada.



Figura 31 – Tela da vitrine de produtos e serviços do e-commerce.



Figura 32 – Passo do tipo “ação” para selecionar o produto na vitrine.

Script gerado e utilizado na integração com o *Selenium*:

1. *Selenium.click*(“//ul[@id='vitrine-container']/li[8]/div/div/a/img”);



Figura 33 – Tela de detalhe do produto selecionado na vitrine.



Figura 34 – Passo do tipo “ação” para adicionar o produto no carrinho.

Script gerado e utilizado na integração com o *Selenium*:

* *Selenium.click(“bt-adicionar-ao-carrinho”);*



Figura 35 – Tela do carrinho de compras.



Figura 36 – Passos do tipo “ação” utilizados no carrinho de compras.

Scripts gerados e utilizados na integração com o *Selenium*:

* *Selenium.click(“aceito-contratos”);*
* *Selenium.click(“//img[@alt='prosseguir']”);*

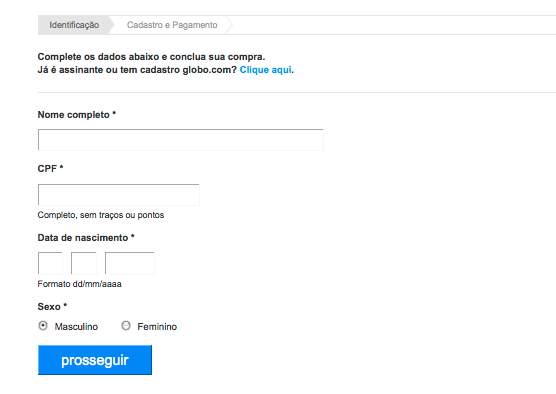


Figura 37 – Tela para informar os dados de identificação.

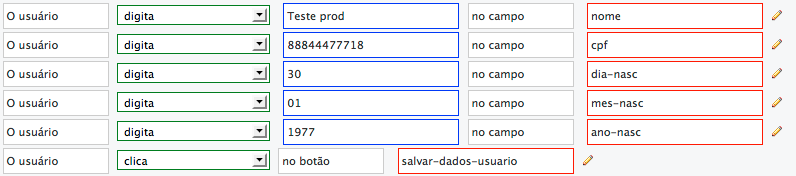


Figura 38 – Passos do tipo “ação” utilizados na tela de identificação.

Scripts gerados e utilizados na integração com o *Selenium*:

* *Selenium.type(“nome”, “Teste prod”);*
* *Selenium.type(“cpf”, “88844477718”);*
* *Selenium.type(“dia-nasc”,”30”);*
* *Selenium.type(“mes-nasc”, “01”);*
* *Selenium.type(“ano-nasc”, “1977”);*
* *Selenium.click(“salvar-dados-usuario”);*

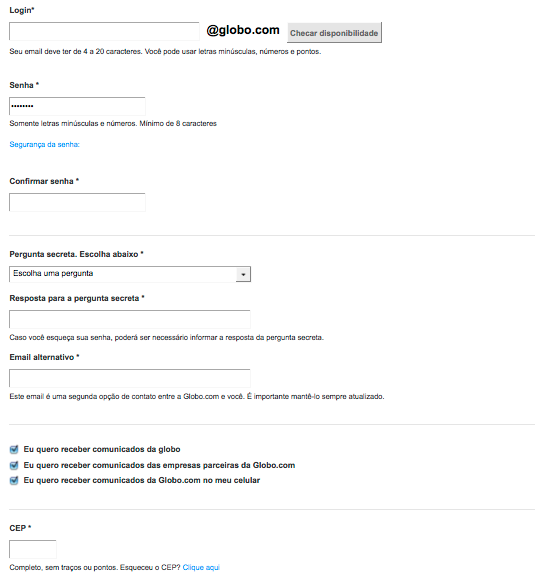


Figura 39 – Tela para informar os dados de cadastro – 1 de 2.

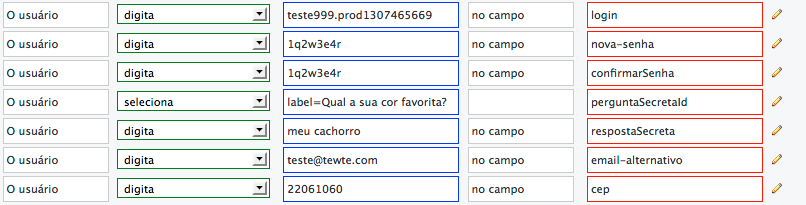


Figura 40 – Passos do tipo “ação” utilizados no tela de cadastro 1 de 2.

Scripts gerados e utilizados na integração com o *Selenium*:

* *Selenium.type(“login”, “teste999.prod1307465669”);*
* *Selenium.type(“nova-senha”, “1q2w3e4r”);*
* *Selenium.type(“confirmarSenha”, “1q2w3e4r”);*
* *Selenium.select(“perguntaSecretaId”, “label=Qual a sua cor favorita?”);*
* *Selenium.type(“respostaSecreta”, “meu cachorro”);*
* *Selenium.type(“email-alternativo”, “teste@tewte.com”);*
* *Selenium.type(“cep”, “22061060”);*

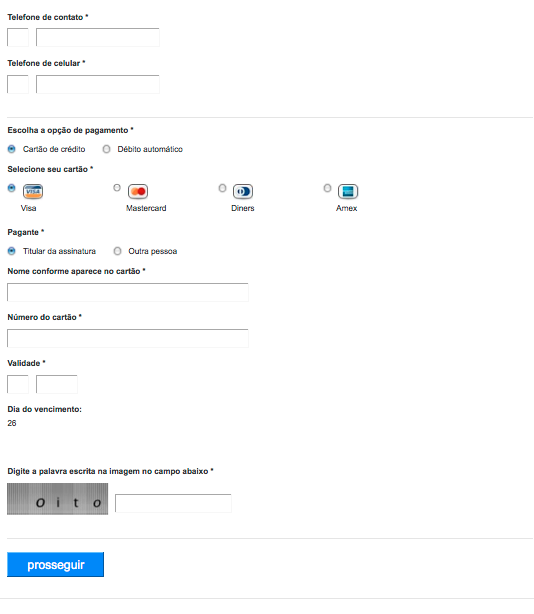


Figura 41 – Tela para informar os dados de cadastro – 2 de 2.

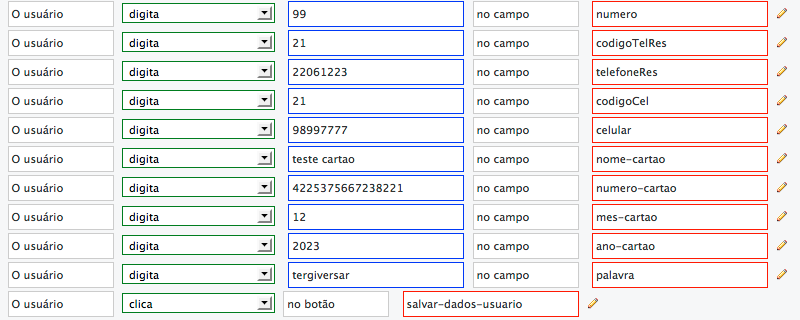


Figura 42 – Passos do tipo “ação” utilizados no tela de cadastro 1 de 2.

Scripts gerados e utilizados na integração com o *Selenium*:

* *Selenium.type(“numero”, “99”);*
* *Selenium.type(“codigoTelRes”, “21”);*
* *Selenium.type(“telefoneRes”, “22061223”);*
* *Selenium.type(“codigoCel”, “21”);*
* *Selenium.type(“telefoneCel”, “98997777”);*
* *Selenium.type(“nome-cartao”, “4225375667238221”);*
* *Selenium.type(“numero-cartao”, “4225375667238221”);*
* *Selenium.type(“mes-cartao”, “12”);*
* *Selenium.type(“ano-cartao”, “2023”);*
* *Selenium.click(“salvar-dados-usuario”);*

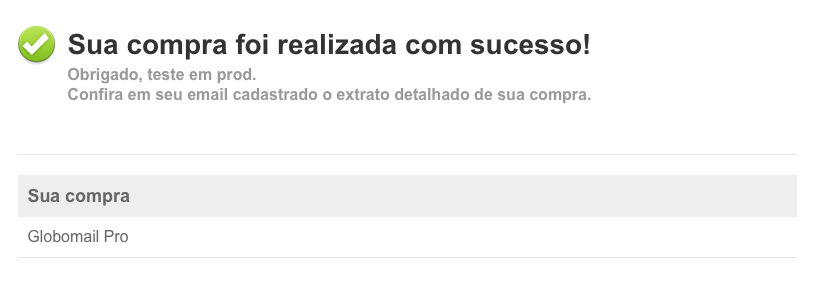


Figura 43 – Tela de sucesso na realização da compra.



Figura 44 – Passo do tipo “verificação” utilizado na tela de sucesso.

Script gerado e utilizado na integração com o *Selenium*:

* *Selenium.isTextPresent?(“Compra realizada com sucesso”)*

## Resultado da execução dos testes

O resultado da geração e execução dos testes é extraído da informação do atributo “status\_test” de cada passo e foi organizado para ser analisado através de três diferentes visões: visão geral, visão do projeto e visão do caso de uso.

Através da visão geral, localizada na página inicial da ferramenta, é possível verificar rapidamente pela lista de projetos, informações sobre a última execução dos testes para cada projeto da lista. A visão exibe as seguintes informações:

**Projeto**: nome do projeto.

**Caso(s) de uso**: quantidade de casos de uso do projeto.

**Qtde. de fluxos**: quantidade de fluxos do projeto.

**Fluxos válidos**: quantidade de fluxos do projeto que executaram os testes corretamente.

**Fluxos Inválidos**: quantidade de fluxo que não executaram os testes corretamente. Neste caso, se a quantidade de fluxos for maior que zero, será apresentado um ícone alertando que neste projeto existem fluxos que não executaram corretamente.

**Última execução**: data da última execução do dos testes.

Conforme apresentado no exemplo da figura 45, na última execução dos testes a ferramenta indicou falha no comportamento de dois fluxos de um projeto.

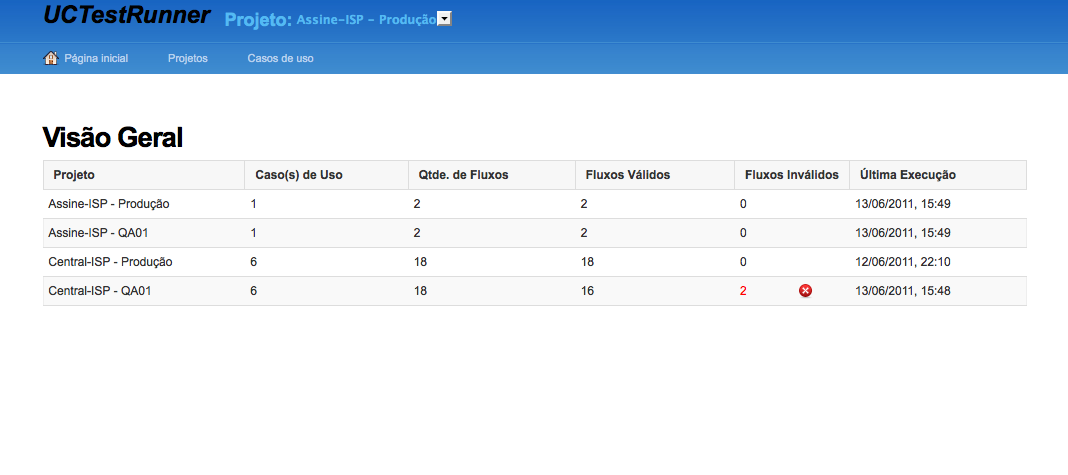


Figura 45 – Visão do resultado da geração e execução dos testes de todos os projetos.

Na visão do projeto, localizada na lista de casos de uso do projeto é possível localizar os casos de uso que apresentaram falha. Conforme exemplo da figura 46 abaixo, dois fluxos de um caso de uso estão indicados como inválidos.

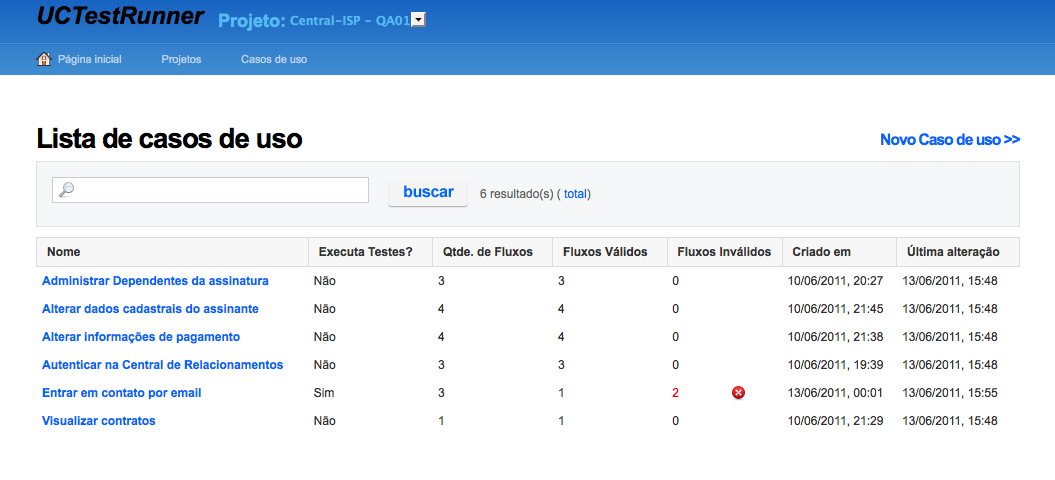


Figura 46 – Visão do resultado da geração e execução dos testes para os casos de uso do projeto.

Na visão do caso de uso, localizada no formulário de cadastro e manutenção de casos de uso, é possível identificar os fluxos e o passo de cada fluxo com comportamento indesejado, conforme exemplo da figura 46 abaixo.



Figura 47 – Visão do resultado da geração e execução dos testes para os fluxos e passos do caso de uso.

Ou seja, para verificar qual passo falhou na geração e execução automática dos testes, é preciso identificar o projeto na página inicial da ferramenta, acessar a lista de casos de uso desse projeto e abrir o formulário do caso de uso. Essas três visões facilitam a identificação e correção do problema.

## Diagrama de implantação

O diagrama da figura 48 abaixo ilustra como a ferramenta será distribuída através de nós de hardware e as suas devidas relações de comunicação.

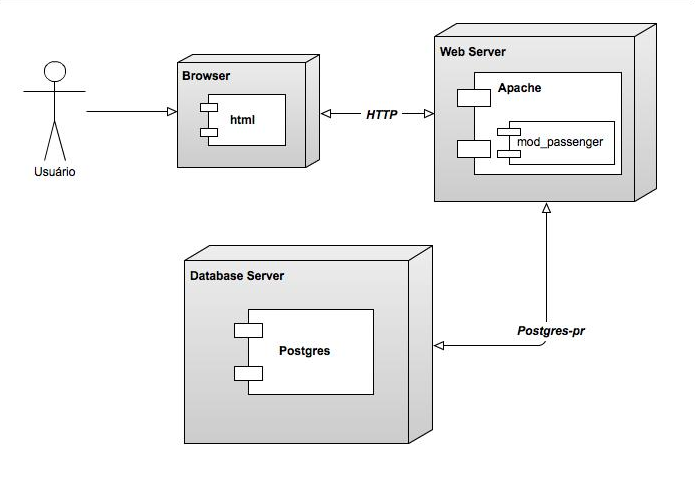


Figura 48 – Diagrama de implantação da ferramenta.

A ferramenta é uma aplicação web servida pelo ***mod\_passenger***, um módulo do Web Server **Apache**, que recebe requisições HTTP através do navegador do usuário e verifica se o pedido deve ser tratado por uma aplicação ***Ruby on Rails***. Se sim, então o ***mod\_passenger*** vai encaminhar o pedido para a aplicação que utiliza o driver ***postgres-pr*** para fazer a comunicação com o banco de dados ***Postgres*** para armazenar e/ou extrair dados e retornar com a resposta para que o navegador possa exibir as informações para o usuário.

## Decisões de projeto

As principais tecnologias descritas abaixo, foram utilizadas para projetar, desenvolver e suportar a ferramenta pois são tecnologias de código aberto, gratuitas, do conhecimento técnico do autor e por se adequarem as práticas utilizadas no contexto da empresa de onde será aplicado o experimento.

***Ruby on Rails*:** *framework* de código aberto escrito na linguagem de programação [*Ruby*](http://pt.wikipedia.org/wiki/Ruby_%28linguagem_de_programa%C3%A7%C3%A3o%29) que promete aumentar velocidade e facilidade no desenvolvimento de sites orientados a banco de dados (*database-driven* web sites), uma vez que é possível criar aplicações com base em estruturas pré-definidas. As aplicações criadas utilizando o framework *Rails* são desenvolvidas com base no padrão de projeto [MVC](http://pt.wikipedia.org/wiki/MVC) (*Model-View-Controller*).

***Aptana Studio*** – ambiente integrado de desenvolvimento de código aberto e gratuito desenvolvido em Java e suporta as linguagens CSS, HTML, pode ser configurado para suportar *Ruby on Rails* através do plug-in *Aptana RedRails*. É baseado no Eclipse, programa similar que por sua vez desenvolve [linguagens de programação](http://pt.wikipedia.org/wiki/Linguagens_de_programa%C3%A7%C3%A3o). É distribuído em múltiplas plataformas ou seja, pode ser utilizado no [Windows](http://pt.wikipedia.org/wiki/Windows), [Linux](http://pt.wikipedia.org/wiki/Linux) e [Mac OS X](http://pt.wikipedia.org/wiki/Mac_OS_X).

**CVS:** CVS ou *Concurrent Version System* (Sistema de Versões Concorrentes) é um sistema de controle de versão de código aberto e gratuito que permite que se trabalhe com diversas versões de arquivos organizados em um diretório e localizados local ou remotamente, mantendo-se suas versões antigas e os logs de quem e quando manipulou os arquivos. É especialmente útil para se controlar versões de um software durante seu desenvolvimento, ou para composição colaborativa de um documento.

***PostgreSQL*:** é um sistema de gerenciamento de banco de dados (SGBD) que utiliza a linguagem SQL Como interface. Desenvolvido como projeto de código aberto.

***Capistrano*:** ferramenta de código aberto e gratuita que permite rodar scripts em múltiplos servidores. Seu principal uso é para realização de *deploy* de aplicações WEB. Com ele é possível automatizar o processo de subida para produção de uma aplicação permitindo a execução de várias tarefas em múltiplos servidores.

***Selenium Remote Control* (RC):** é uma ferramenta que permite automatizar testes em aplicações web. É dividido em duas partes:

1. Um servidor que automaticamente inicia e encerra o navegador web, e atua como um proxy HTTP para solicitações web provindas desses navegadores.
2. Biblioteca cliente para montar e executar os scripts de teste. Disponível para diferentes linguagens de programação.

# Estudo de caso

Neste capítulo serão apresentados os sistemas utilizados na aplicação da abordagem proposta, bem como a tecnologia em que foram desenvolvidos, o escopo do experimento e a situação atual da documentação e dos testes. Também serão apresentados os resultados obtidos na aplicação do estudo de caso.

## Visão geral das aplicações utilizadas nos testes

Para realizar o estudo de caso foram utilizadas duas aplicações web: o *Assine Já* e a *Central de Relacionamentos*.

O *Assine Já* é um web site desenvolvido para suportar o comércio eletrônico dos produtos e serviços de um grande provedor da internet. Em seu funcionamento, um usuário da internet acessa o site do *Assine Já*, adiciona os produtos e/ou serviços de seu interesse em um carrinho de compras, lê e aceita os termos contratuais, informa os dados cadastrais, os dados de pagamento e finaliza o cadastro. Ao finalizar o procedimento, o sistema cadastra os dados do usuário que, a partir desse momento, passa a ser um assinante do provedor e ter direito a usar os produtos e serviços comprados.

A *Central de Relacionamentos* é outro web site desenvolvido para permitir que o usuário ao se cadastrar pelo *Assine Já* e se tornar um assinante, possa visualizar os produtos e serviços comprados, visualizar e editar seus dados cadastrais, suas informações de pagamento, visualizar suas cobranças, regularizar sua situação financeira, administrar seus dependentes, visualizar o contrato dos produtos, visualizar o histórico das suas conexões na internet através do provedor e entrar em contato para tirar dúvidas, relatar problemas e realizar reclamações, através da opção de envio de e-mail ou através mensagens instantâneas.

As duas aplicações fazem parte da camada visão dentro da infra estrutura de projetos da plataforma de vendas do provedor e foram desenvolvidas utilizando as seguintes tecnologias:

* ***Java***: linguagem de programação orientada a objetos;
* ***JSP***: *JavaServer Pages*, tecnologia utilizada no desenvolvimento de aplicações para Web;
* ***Struts***: *framework* para auxiliar no desenvolvimento da camada controladora numa estrutura seguindo o padrão Modelo 2;
* **Modelo 2**: arquitetura web que utiliza o projeto MVC (*Model-View-Controller*) para separar a lógica de negócio da lógica de apresentação do conteúdo, permitindo o desenvolvimento, teste e manutenção isolado de ambos;
* ***Ajax***: *Asynchronous JavaScript and XML*, desenvolvimento de aplicações em *JavaScript* com *XML*, providas por navegadores, para tornar páginas Web mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações;
* ***CSS***: *Cascading Style Sheets,* linguagem de estilo que define a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento;
* ***JavaScript***: linguagem de programação baseada na linguagem de programação *ECMAScript* padronizada pela Ecma *international* nas especificações **ECMA-262** e ISO/IEC 16262 e é atualmente a principal linguagem para programação *Client-Side* em navegadores web;
* ***Apache Tomcat***: servidor web Java, mais especificamente, um container de *servlets*. Possui algumas características próprias de um servidor de aplicação, porém não pode ser considerado um servidor de aplicação por não preencher todos os requisitos necessários.

O *Assine Já* e a *Central de Relacionamentos* não fazem comunicação direta com o banco de dados para armazenar e consultar informações, conforme ilustrado na figura 49 abaixo. Existe entre as aplicações e o banco de dados outra aplicação mediadora chamada *Kernel* onde estão concentradas as regras de negócio do provedor. Construído com a tecnologia EJB - *Enterprise JavaBeans* um dos principais componentes da plataforma J2EE - *Java 2 Enterprise Edition*.

O *kernel* é uma aplicação do tipo servidor que executa no container do servidor de aplicação e seu principal objetivo é fornecer um rápido e simplificado desenvolvimento de aplicações Java baseado em componentes distribuídos, transacionais, seguros e portáveis.

Ao utilizar uma das aplicações, os usuários de um dispositivo com conexão a internet e através de um navegador web, interage com o *Assine Já* ou com a *Central de Relacionamentos*, que valida e formata os dados informados e as envia para o *Kernel*. O *Kernel* recebe a requisição, processa as validações de negócio, armazena e obtêm informações no banco de dados para retornar as aplicações clientes, o *Assine Já* ou e a *Central de Relacionamentos*, que tratam as informações e as apresentam ao usuário.

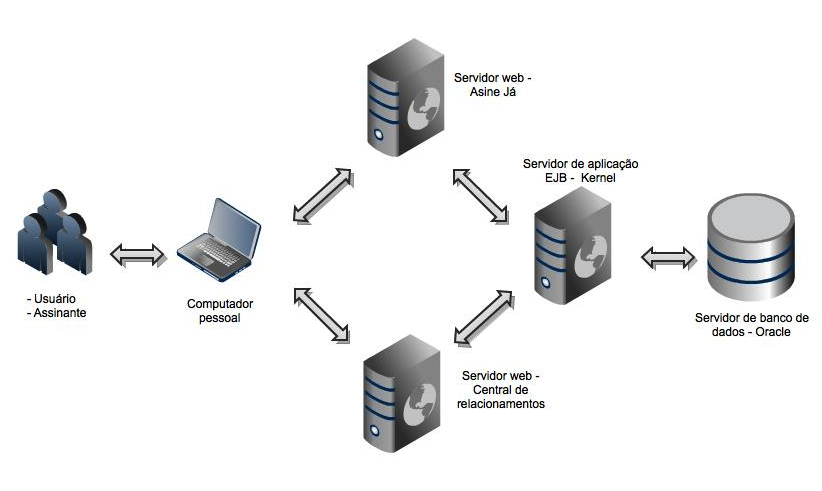


Figura 49 – Visão geral das aplicações utilizadas no experimento.

As aplicações tem forte dependência de negócio e são maduras em produção e consequentemente sofrem constantes manutenções para melhoria e otimização do software desenvolvido, como também reparo de defeitos. Atualmente o *Assine Já* é utilizado por aproximadamente quatro mil usuários diariamente, sendo que desses quatro mil, aproximadamente três mil são convertidos em assinatura para o provedor. A *Central de Relacionamentos* prove suporte para que aproximadamente um milhão de assinantes possa interagir com as informações da sua assinatura no provedor.

## Escopo do experimento

Dado o elevado número de elementos que podem ser cadastrados no sistema e o tempo disponível para a realização dos testes, apenas um subconjunto de elementos foi selecionado pela equipe de desenvolvimento para passar pelo processo de geração de testes proposto e para comparar com outras formas de teste. Cada funcionalidade abaixo pode ser expressa em um caso de uso e poderão também ser referenciadas como tal a partir de agora.

Funcionalidade do *Assine Já* e seus principais fluxos de eventos utilizados no estudo de caso:

* UC1 - Efetuar compra
* Efetuar primeira compra - produto Banda Larga - Cartão de Crédito;
* Efetuar primeira compra - produto Banda Larga - Débito Instantâneo;
* Efetuar segunda compra - produto Banda Larga - cartão de Crédito;
* Efetuar segunda compra - produto Banda Larga - Débito Instantâneo;

Funcionalidades da *Central de Relacionamentos* e seus principais fluxos de eventos utilizados no estudo de caso:

* UC2 – Realizar autenticação
* Autenticar com sucesso;
* Autenticar com usuário incorreto;
* Autenticar com senha incorreta;
* UC3 - Alterar dados cadastrais
* Alterar informações de cadastro;
* Alterar senha;
* Alterar informações de localização;
* Alterar contatos telefônicos;
* UC4 - Administrar dependentes
* Adicionar dependente;
* Alterar dependente;
* Remover dependente;
* UC5 - Alterar informações de pagamento;
* Alterar dados do cartão de crédito;
* Alterar para débito instantâneo;
* Alterar dados do Débito instantâneo;

O caso de uso “UC1 - Efetuar compra” do *Assine Já,* cobre os principais fluxos de compra da aplicação, que é o principal canal de entrada de novas assinaturas no provedor, aproximadamente 80% dos assinantes cadastram suas assinaturas utilizando esses fluxos. Os quatro casos de uso da *Central de Relacionamentos* foram considerados pela equipe como as funcionalidades mais relevantes aos assinantes do provedor.

A indisponibilidade ou o mau funcionamento em umas das funcionalidades citadas acima, tanto do *Assine Já* quanto da *Central de Relacionamentos*, pode aumentar o número de chamadas telefônicas no *Call Center,* aumentando o TMA (Tempo Médio de Atendimento) dos atendentes, o que além de aumentar o custo da operacional, impacta o cadastro de novas assinaturas, já que o *Call Center*, assim como o *Assine Já*, é outro canal de venda.

Ao acionar a execução dos testes automáticos e verificar que as funcionalidades dos casos de uso executaram corretamente, a equipe estará segura de que as manutenções efetuadas no *Assine Já,* na *Central de Relacionamentos* ou no *Kernel* não causaram impacto nas funcionalidades e que elas estão em conformidade com a documentação descrita nos fluxos de evento do caso de uso.

## Situação atual

Atualmente, as aplicações acima descritas não dispõem de nenhuma documentação formal de requisitos, em consequência disto, não existem documentos de testes extraídos de documentos funcionais. As aplicações não são cobertas por testes funcionais automatizados e os testes são executados de forma manual. A cada desenvolvimento de uma manutenção em uma das aplicações, seja ela corretiva ou evolutiva, existe a necessidade de realizar testes de regressão nas funcionalidades das aplicações alteradas, uma tarefa que exige muito tempo e esforço da equipe de desenvolvimento. Os testes manuais são executados de maneira desordenada, o que não garante um resultado único a cada execução.

## Resultados obtidos

Para avaliar os resultados obtidos através da aplicação da abordagem proposta em um cenário real na fábrica de software do um provedor, foram realizados as seguintes medições e métodos:

### Esforço para redigir os casos de uso

A primeira atividade realizada na aplicação do experimento foi medir o esforço para redigir cada caso de uso, listado no item “5.2.Escopo do experimento”, utilizando a ferramenta proposta. Essa medição servirá como base para calcular o esforço de redigir todos os fluxos identificados e para a comparação com outros métodos de teste. O esforço para cadastrar cada caso de uso engloba os dois passos seguintes:

* **Identificar os elementos do HTML** – tempo em minutos para identificar os elementos do HTML. Dado que o experimento foi executado utilizando aplicações já desenvolvidas e a ferramenta proposta gera os scripts para integrar ao *Selenium* e executar as “ações” e “verificações” no formulários web, foi necessário fazer um levantamento dos elementos HTML de cada formulário web utilizado no campo “alvo” de cada passo dos fluxos descritos no casos de uso. A forma mais ágil encontrada para identificar os elementos foi utilizar o *Selenium IDE* para navegar por cada fluxo do caso de uso antes de documentá-lo.
* **Redigir os fluxos de eventos** - tempo em minutos para redigir o caso de uso e seus fluxos de eventos (principal e alternativos).

O esforço para realizar cada atividade foi cronometrado e depois somados para chegar ao tempo total de cadastro do caso de uso, conforme tabela 4 abaixo. As medições da tabela 4 foram utilizadas como base para a estimativa do item 5.4.2 e para as comparações realizadas nos itens 5.4.3 e 5.4.4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Casos de uso | Quantidade de Fluxos | Identificar HTML  (minutos) | Redigir casos de uso (minutos) | Total  (minutos) |
| UC1 | 4 | 4.92 | 38.52 | 43.44 |
| UC2 | 3 | 1.50 | 6.96 | 8.46 |
| UC3 | 4 | 4.80 | 12.60 | 17.40 |
| UC4 | 3 | 5.65 | 14.91 | 20.56 |
| UC5 | 3 | 2.57 | 10.58 | 13.15 |
| Total (minutos) | 17 | 19.44 | 83.57 | 103.01 |

Tabela 4 – Esforço para redigir os casos de uso.

### Esforço para assegurar que tenham sido criados suficientes scripts de teste;

Para medir o esforço de cadastrar a documentação completa das aplicações *Assine Já* e *Central de Relacionamentos* na ferramenta proposta e assegurar que os testes cubram o maior número de funcionalidades e regras possíveis para dar mais confiança a equipe de desenvolvimento e para outros interessados, foram realizadas as seguintes atividades:

1. **Identificar o número máximo de fluxos** – identificar todos os possíveis fluxos das duas aplicações que não foram utilizados no escopo do experimento. Para identificar os fluxos, foi feita uma navegação exploratória nas aplicações. Abaixo, os outros fluxos identificados para cada caso de uso:

* Novos fluxos identificados no *Assine Já*:
* UC1 - Efetuar compra
  + Efetuar primeira compra - produto Acesso Discado - Cartão de Crédito;
  + Efetuar primeira compra - produto Acesso Discado - Débito Instantâneo;
  + Efetuar segunda compra - produto Acesso Discado - cartão de Crédito;
  + Efetuar segunda compra - produto Acesso Discado - Débito Instantâneo;
  + Efetuar primeira compra - produto Avulso - Cartão de Crédito;
  + Efetuar primeira compra - produto Avulso - Débito Instantâneo;
  + Efetuar segunda compra - produto Avulso - cartão de Crédito;
  + Efetuar segunda compra - produto Avulso - Débito Instantâneo;
  + Efetuar primeira compra – CPF Inválido;
  + Efetuar primeira compra – data de nascimento inválida;
  + Efetuar primeira compra – usuário inválido;
  + Efetuar primeira compra – senha inválida;
  + Efetuar primeira compra – confirmação de senha inválida;
  + Efetuar primeira compra – e-mail inválido;
  + Efetuar primeira compra - Cartão de Crédito – numero do cartão inválido;
  + Efetuar primeira compra - Cartão de Crédito – data vencimento do cartão inválida;
  + Efetuar primeira compra - Cartão de Crédito – CPF outro pagante inválido;
  + Efetuar primeira compra - Cartão de Crédito – data nascimento do pagante inválida;
  + Efetuar primeira compra - Débito instantâneo – dados bancários inválidos;
  + Efetuar primeira compra - Débito instantâneo – CPF outro pagante inválido;
  + Efetuar primeira compra - Débito instantâneo – data nascimento do outro pagante inválida;
  + Efetuar primeira compra – palavra de segurança não confere;
* Novos fluxos identificados na *Central de Relacionamentos*:
* UC2 – Realizar autenticação
  + Autenticar com senha e usuário inválido;
  + Autenticar com usuário inadimplente;
  + Autenticar com usuário bloqueado;
  + Autenticar com usuário com data do cartão de crédito expirada;
* UC3 - Alterar dados cadastrais
  + Alterar informações de cadastro – CPF inválido;
  + Alterar informações de cadastro – e-mail inválido;
  + Alterar informações de cadastro – data nascimento inválida;
  + Alterar senha – senha atual errada;
  + Alterar senha – nova senha inválida;
  + Alterar senha – confirmação de senha inválida;
  + Alterar informações de localização – CEP inválido;
  + Alterar contatos telefônicos – telefone contato inválido;
  + Alterar contatos telefônicos – celular inválido;
* UC4 - Administrar dependentes
  + Adicionar dependente – dependente existente;
  + Adicionar dependente – senha inválida;
  + Adicionar dependente – senha não confere;
  + Adicionar dependente – e-mail inválido;
  + Adicionar dependente – celular inválido;
* UC5 - Alterar informações de pagamento
  + Alterar dados do cartão de crédito – numero cartão inválido;
  + Alterar dados do cartão de crédito – data de validade do cartão inválida;
  + Alterar dados do cartão de crédito – CPF do outro pagante inválido;
  + Alterar dados do cartão de crédito – data nascimento do outro pagante inválida;
  + Alterar dados do Débito instantâneo – dados bancários inválidos;
  + Alterar dados do Débito instantâneo – CPF outro pagante inválido;
  + Alterar dados do Débito instantâneo – data nascimento do outro pagante inválida;
* UC6 – Visualizar contratos
  + Visualizar contrato de um produto;
  + Visualizar contratos de mais de um produto;
  + Visualizar impressão;
* UC7 – Visualizar histórico de conexões
  + Visualizar histórico acesso banda larga;
  + Visualizar histórico acesso discado;
  + Visualizar impressão;
* UC7 – Baixar discador
  + Pesquisar cidades;
  + Baixar discador;
  + Acessar Central de Ajuda;
* UC8 – Entrar em contato
  + Entrar em contato via chat;
  + Entrar em contato por e-mail;
  + Entrar em contato por e-mail para cancelamento;
  + Entrar em contato por telefone;
  + Acessar Central de Ajuda;

1. **Calcular o tempo médio para cadastrar cada caso de teste** – calcular o tempo médio para redigir cada caso de teste na ferramenta proposta, tomando com base o tempo total para cadastrar os casos de uso utilizados no escopo do experimento, dividido pelo número de fluxos, ambos resultados da tabela 4. Ou seja, 103.01 minutos (tempo total) / 17 (número de fluxos cadastrados) = 6.06 minutos (tempo médio encontrado para cadastrar cada fluxo).
2. **Calcular o esforço para redigir cada caso de uso** - Calcular o tempo para redigir os todos os fluxos e cada caso de uso, tanto os utilizados no experimento, quanto os identificados. Como exemplo, podemos utilizar o cálculo realizado para o caso de uso UC1. Foram redigidos na ferramenta proposta 4 fluxos que somados aos outros 22 fluxos identificados, totalizam 26 fluxos. Esses 26 fluxos foram multiplicados por 6.06 minutos (tempo médio para cadastrar um fluxo) e o resultado para cadastrar todos os fluxos do UC1 foi de 157.54 minutos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Casos de uso | Principais fluxos cadastrados | Outros fluxos identificados | Total de fluxos | Tempo médio de cadastro do fluxo  (minutos) | Total  (minutos) |
| UC1 | 4 | 22 | 26 | 6.06 | 157.54 |
| UC2 | 3 | 4 | 7 | 6.06 | 42.42 |
| UC3 | 4 | 9 | 13 | 6.06 | 78.77 |
| UC4 | 3 | 5 | 8 | 6.06 | 48.48 |
| UC5 | 3 | 7 | 10 | 6.06 | 60.59 |
| UC6 | 0 | 3 | 3 | 6.06 | 18.18 |
| UC7 | 0 | 3 | 3 | 6.06 | 18.18 |
| UC8 | 0 | 4 | 4 | 6.06 | 24.24 |
| Total | 17 | 57 | 74 |  | 448.40 |

Tabela 5 – Estimativa do esforço para redigir os casos de teste identificados.

1. **Calcular o esforço para redigir todos os fluxos identificados –** foram identificados um total de 74 fluxos que multiplicados por 6.06 minutos (tempo médio para cadastrar cada fluxo) totalizam 448.40 minutos.

O valor aproximado de 448.40 minutos é tempo total estimado para redigir todos os 74 fluxos identificados como necessários para assegurar que a aplicação gere e execute scripts de testes suficientes para dar confiança aos envolvidos com a aplicação que seu comportamento está conforme esperado. Para cadastrar os 17 fluxos utilizados no escopo deste experimento foram gastos 103.01 minutos (ver tabela 4), ou seja, foram redigidos apenas 23% do total de fluxos identificados para realizar o experimento.

### Comparação com testes produzidos manualmente

Diante do fato da inexistência de testes funcionais automatizados nas aplicações utilizadas no experimento, do conhecimento adquirido no *framework* de testes *Selenium* e da tecnologia que as aplicações foram desenvolvidas, foi utilizado *JUnit* (JUnit, 2011) e *Selenium* para codificar manualmente os testes dos casos de uso. É importante ressaltar que a escrita manual do teste não teve o auxílio de funcionalidades de criação de código presentes em algumas *IDE’s* e nem uso de “copiar” e “colar”.

Para a criação dos testes com codificação manual foi utilizada a IDE Eclipse (Eclipse, 2011). Cada caso de uso codificado manualmente consistiu em um arquivo Java (Ex.: EfetuarCompraTest.java) de execução automatizada através de JUnit e a interação com o navegador automatizada a partir da API do *Selenium* para Java (Selenium, 2011). Foram criados manualmente 5 classes com 17 métodos para simular o mesmo cadastro realizado através da ferramenta proposta, conforme tabela 4.

A figura 50, a figura 51 e a figura 52 abaixo ilustram a estrutura criada para suportar a codificação manual e um exemplo do código criado para testar o fluxo principal do “UC1 - Efetuar Compra”.



Figura 50 – Visão testes criados manualmente através do Eclipse IDE (Eclipse, 2011).



Figura 51 – Código *JUnit* para invocar a execução dos testes criados manualmente.



Figura 52 – Exemplo de um caso de teste criado manualmente utilizando Java com *Selenium* *API*.

O tempo gasto na codificação manual foi comparado com o tempo gasto para escrever o caso de uso apoiado pela ferramenta. O tempo para escrita dos oráculos não foi levado em conta em nenhum dos casos, pois, em ambos os testes, teriam que ser escritos os mesmos oráculos.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Geração manual | UC1  (minutos) | UC2  (minutos) | UC3  (minutos) | UC4  (minutos) | UC5  (minutos) | Total |
| Criar cada caso de uso | 74.50 | 13.67 | 28.86 | 56.89 | 18.15 | 192.70 |
| Refatorar o teste | 21.40 | 5.79 | 9.78 | 15.78 | 7.65 | 60.40 |
| Tempo total (minutos) | 95.9 | 19.46 | 38.64 | 72.67 | 25.8 | 252.47 |

Tabela 6 – Esforço para criar os casos de teste manualmente.

Enquanto que para o cadastro de 5 casos de uso com 17 fluxos foram gastos apenas 103.01 minutos utilizando a ferramenta proposta (conforme tabela 4), a codificação manual consumiu 252.47 minutos, conforme tabela 6 acima. Assim, a geração do script de teste com o uso da ferramenta proposta neste trabalho exigiu, aproximadamente, apenas 40.8% do tempo gasto na geração manual do script.

### Comparação com testes produzidos através de “*capture and replay*”

A abordagem para a geração e execução automática de testes apresentada neste trabalho e a abordagem “*capture and replay*” foram comparadas quanto ao tempo gasto para a geração dos testes. Essa comparação deu-se através dos testes gerados para cinco casos de uso, um do *Assine Já* e quatro da *Central de Relacionamentos.* A ferramenta escolhida para gerar os casos de teste a partir de “*capture and replay*” foi o *Selenium IDE,* uma extensão do navegador *Mozilla Firefox 3*.6.

A geração dos scripts de teste com *Selenium IDE* engloba três passos:

1. **Gravação:** gravar as ações do usuário para cada caso de uso.
2. **Refatoração do código:** todos os passos do caso de uso foram gravados em uma única execução do *Selenium IDE*, o que necessitou que o código gerado fosse refatorado para atribuir uma função a cada caso de teste e uma função inicial que prepara o ambiente para cada caso de teste (Todos estes passos encontravam-se em apenas uma única função no código oriundo da gravação). Ao executar os testes gravados pelo próprio *Selenium IDE*, foram encontrados problemas na execução de alguns passos, principalmente quando o formulário web utiliza *AJAX* (***A****synchronous* ***J****avascript* ***a****nd* ***X****ML*) e esses também foram refatorados para que o testes executassem corretamente.
3. **Inserção de oráculos**: mesmo com “*capture and replay*” ainda é necessário inserir as chamadas para as funções que exercem o papel de oráculos (ou assertivas).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Selenium IDE*  *(capture and replay*) | UC1  (minutos) | UC2  (minutos) | UC3  (minutos) | UC4  (minutos) | UC5  (minutos) | Total  (minutos) |
| Gravação do fluxo | 17.74 | 4.25 | 7.69 | 7.23 | 5.78 | 42.69 |
| Refatoração do código | 23.34 | 3.34 | 9.35 | 11.88 | 7.5 | 55.41 |
| Inserção de oráculos | 11.89 | 2.89 | 6.3 | 8.5 | 4.7 | 34.28 |
| Tempo total em minutos | 52.97 | 10.48 | 23.34 | 27.61 | 17.98 | 132.38 |

Tabela 7 – Esforço criar os casos de teste através da técnica “*capture and replay*”.

Como podemos ver a partir dos dados da tabela acima comparada com o resultado da tabela 4, o ganho de tempo na geração de testes com a metodologia presente neste trabalho em relação ao tempo gasto para a geração de testes utilizando “*capture and replay*” foi de, aproximadamente, 28,4% para os mesmos 5 casos de uso e 17 fluxos.

### Dificuldades encontradas

Um problema encontrado diz respeito a identificação dos elementos do *HTML* das aplicações testadas na hora de redigir o caso de uso. Uma vez que a ferramenta proposta utiliza a integração com o *framework Selenium* para executar os scripts de teste na interface web, ele precisa conhecer o elemento “alvo” no formulário *HTML* para executar uma “ação” ou “verificação”. Então, antes de redigir cada caso de uso, foi necessário identificar cada elemento do *HTML* do formulário web para que fosse possível redigir cada passo do tipo “ação” e “verificação” do fluxos de eventos. Do tempo total de 103.01 minutos gastos no cadastro dos 17 fluxos utilizados no experimento, 19.44 minutos foram utilizados para identificar os elementos *HTML*. Ou seja, aproximadamente 20% do tempo total para redigir os casos de uso foram utilizados na atividade. Atividade que não seria necessária se o experimento estivesse sido realizado em aplicações ainda não implementadas, uma vez que ao documentar o caso de uso e redigir cada passo do tipo “ação” e “verificação”, os campos “alvo” seriam sugeridos pelo redator e posteriormente utilizados pelo desenvolvedor na implementação do formulário web da aplicação.

Outra dificuldade encontrada foi que o formulário de cadastro dos casos de uso se mostrou muito extenso e cansativo de ler. O problema pode ser exemplificado no caso de uso “UC1”, onde foram identificados um total de 26 fluxos e eventos e todos são semelhantes ao fluxo principal ilustrado na figura 53 abaixo. Cadastrar os 26 fluxos tornaria o formulário demasiadamente grande e difícil de ler. Tal problema poderia ser minimizado se fosse implementado uma forma de reaproveitar fluxos ou passos semelhantes entre os fluxos e assim diminuir o número de passos repetidos do fluxo de eventos e consequentemente o seu tamanho.



Figura 53 – Fluxo básico de eventos do caso de uso UC1.

# Conclusões e trabalhos futuros

O principal objetivo deste trabalho foi pesquisar na literatura maneiras mais eficazes para facilitar e incentivar a documentação de requisitos testáveis e também minimizar o esforço e as dificuldades para automatizar testes funcionais em aplicações web. Ambos problemas observados pelo autor, juntamente com outros integrantes de uma equipe de software, que há três anos adotou a metodologia de desenvolvimento de software ágil ante a metodologia tradicional.

Através das pesquisas e estudos realizados, encontramos no caso de uso um instrumento capaz de apoiar e facilitar a equipe na elicitação e geração dos testes. Propôs-se utilizar uma abordagem de casos de uso direcionados por comportamento para documentar requisitos de software e, a partir dessas informações, automaticamente gerar e executar scripts de teste para verificar se comportamento funcional de aplicações web estão em conformidade com o especificado. Foi escolhida essa abordagem limitada dos casos de uso para facilitar na construção de uma ferramenta.

Para isso foi especificado, projetado e desenvolvido uma ferramenta para viabilizar a documentação de requisitos funcionais na forma de casos de uso, que pudesse ser utilizada mesmo por pessoas com pouco ou nenhum treinamento em computação, bem como utilizar dados dessa documentação para gerar e executar automaticamente scripts de testes e verificar se o comportamento da aplicações web estão em conformidade que o que foi descrito. A ferramenta desenvolvida disponibiliza um formulário que permite realizar o cadastro de casos de uso e depois utiliza as informações cadastradas, em especial dos fluxos de eventos (principal e alternativos), para gerar e executar scripts de teste utilizando a integração com a ferramenta *Selenium* (Selenium, 2011), responsável pela interação com os elementos da interface web.

A avaliação da eficácia do procedimento e da ferramenta deu-se através da aplicação destes na geração de testes funcionais para dois sistemas reais e através de comparação com técnicas manuais e técnicas tradicionais de geração de testes automatizados aplicadas ao mesmo sistema.

De acordo com os resultados obtidos, os testes gerados automaticamente com o auxílio da ferramenta apresentaram as seguintes vantagens:

* Pouco esforço para redigir todos os fluxos de eventos identificados e assegurar que foram gerados casos de teste suficientes para verificar o comportamento da aplicação web;
* Redução do tempo gasto para gerar testes em comparação com a geração manual;
* Redução no tempo necessário para gerar os testes em comparação com a abordagem *“capture and replay”* utilizando-se *Selenium IDE*;
* Minimizou o problema da falta de documentação de requisitos;
* Melhor detalhamento dos requisitos utilizando casos de uso.
* Documentação que gera testes ao invés de histórias.

Como desvantagens, podemos citar a dificuldade em identificar os objetos do formulário web na hora de digitar os passos, o que prolonga o tempo de escrita caso de uso, e testes com fluxos longos, com muitos passos, pode gerar um caso de uso muito extenso e cansativo para ler. Outra desvantagem foi a ausência de um critério mais adequado na escolha dos casos de teste o que pode levar a testes insuficientes.

Assim, acreditamos, perante os resultados observados, que a abordagem apresentada é uma boa alternativa para documentar requisitos, como também para gerar testes funcionais em aplicações web se comparada às técnicas mais utilizadas para geração e execução de testes funcionais, consumindo um tempo menor que as demais técnicas testadas. A abordagem apresentada possibilitou aos usuários uma maneira mais fácil para criar e alterar os testes, uma vez que a ferramenta encapsula os conhecimentos em programação, exigidos na técnica de codificação manual, como também os conhecimentos técnicos para refatorar e inserir oráculos, exigidos na técnica de “*capture and replay*”.

Este trabalho permitiu também fazer uma análise detalhada da ferramenta *Selenium* (Selenium, 2011) e dos recursos oferecidos por ela para o desenvolvimento de scripts para automação de testes. Mesmo com todas as deficiências, o *Selenium* mostrou-se ser uma plataforma com bastante potencial para o desenvolvimento de scripts de teste, com suporte a vários recursos, e a possibilidade de extensão de diversas funcionalidades bastante interessantes.

Por fim, os trabalhos futuros sobre a ferramenta podem amenizar algumas dificuldades encontradas nesta abordagem e diminuir ainda mais o esforço necessário para documentar os requisitos e como consequência, ganhar mais agilidade na geração e execução de testes.

## Trabalhos futuros

Para dar continuidade a este trabalho, podemos identificar as seguintes melhorias na ferramenta proposta:

* Implementar uma funcionalidade para identificar os elementos do formulário *HTML* para o caso de aplicações web já implementadas. A identificação de objetos de interface gráfica continua sendo uma tarefa muito difícil que diminui consideravelmente a viabilidade e rentabilidade da automação de testes.
* Implementar o reaproveitamento de fluxos de eventos de outros casos de uso para diminuir a quantidade de passos em cada fluxo e assim diminuir o tempo de cadastro. Como exemplo, podemos citar o fluxo de eventos “Autenticar com sucesso” do caso de uso “Autenticar na central de relacionamentos” que poderia ser utilizado em todos os outros fluxos da aplicação desse aplicação.
* Implementar a utilização de um banco de dados específico de testes para as aplicações homologadas, onde a ferramenta possa preparar a massa de dados de teste de cada aplicação web, popular o banco com essas informações e no fim da execução do teste realizar a limpeza dos dados. Evitando a possibilidade de erros provindos de informações erradas.
* Melhorar o leiaute do formulário de cadastro de caso de uso e eliminar alguns itens desnecessários para deixar os documentos mais enxutos e interessantes;
* Enfim, realizar novos estudos de caso com diferentes aplicações e uma massa maior de dados para verificar se as melhorias propostas melhoram o processo.

# Bibliografia

1. Agile Manifesto web site**;** Disponível em: <http://agilemanifesto.org/>. Acesso em: 09 fev. 2010.
2. Alhir, S. S. **Learning UML**, O'Reilly & Amp; Associates, Inc. 2003.
3. Ammann, P. and Offutt, J. **Introduction to Software Testing**. New York, Cambridge University Press, 2008.
4. Araújo, T.P.; Staa, A.v.; **Um Método Baseado em Comportamento com Foco no Desenvolvimento de Aplicações Baseadas em Interfaces Gráficas**. Monografias em Ciência da Computação 26/09; Departamento de Informática, Pontifícia Universidade Católica (PUC-Rio), Rio de Janeiro, RJ; 2009.
5. Balcer, M. J. **Category-Partition Method**. Communications of the ACM. 676-686, 1988.
6. Beck, K.; Andres, C. **Extreme Programming Explained: Embrace Change**. Second Edition. Addison-Wesley Professional, 2004.
7. Berg van den, K.; Mehmet; et al. **Use Cases in Object-Oriented Software Development**. Enschede, Universiteit Twente, AMIDST/CTIT, 1999.
8. Bertolino, A. **Software Testing Research: Achievements, Challenges, Dreams**. Future of Software Engineering, IEEE Computer Society, 2007.
9. Binder, R.V. **Testing Object-Oriented Systems: Models, Patterns, and Tools**. Addison-Wesley Professional, 1999.
10. Caldeira, L. **Geração semi-automática de massas de testes funcionais a partir da composição de casos de uso e tabelas de decisão**. Rio de Janeiro, 2010, 87p. Dissertação de Mestrado em Informática. PUC-RIO.
11. Cockburn, A.; **Wrinting Effective Use Cases**. Addison-Wesley Professional, 2000. 304 p.
12. Cockburn, A.; Adolph, S.; Bramble, P.; Pols, A. **Patterns** **for Effective Use Cases**, Addison-Wesley, 2002. 272 p.
13. Dranidis, D.; Tigka , K.; et al. **Formal modelling of use cases with X-machines**. SEEFM'03 - 1st South-East European Workshop on Formal Methods, Thessaloniki, Greece, 2003.
14. Dustin, Elfriede.; Garret, Thon.; Gauf, Bernie. **Implementing automated software testing : how to save time and lower costs while raising quality**. Addison Wesley Professional, 2009 - 368 p. ISBN 0-321-58051-6.
15. Eclipse web site; Disponível em: < http://eclipse.org/>. Acesso em: 02 jun. 2011.
16. Fröhlich, P. and Link, J. **Automated Test Case Generation from Dynamic Models**. Proceedings of the 14th European Conference on Object-Oriented Programming, Springer-Verlag, 2000.
17. Gutiérrez, J.J.; Escalona, M.J.; Mejías, M. and Torres, J. **Derivation of Test Objectives Automatically**, Proc. 15th Int"l Conf. Information Systems Development, 2006.
18. Heumann, J. **Generating Test Cases from Use Cases**. The Rational Edge ezine; New York, NY: International Business Machines; 2001. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf. Acesso em: 10 mar. 2011.
19. Highsmith, J. **Agile Project Management: Creating Innovative Products, Second Edition**. Addison-Wesley, 2009
20. Jacobson, I.; et al. **Object-Oriented Software Engineering: A Use Case Driven Approach**. Reading, Mass.: Addison-Wesley, 1992.
21. Jacobson, I; **Object-Oriented Software Engineering: A Use Case Driven Approach**. Addison-Wesley Professional, 1992.
22. Jacobson, I. **Use cases - Yesterday, today, and tomorrow.** Software and System Modeling 3(3): 210-220, 2004.
23. JBehave web site; Disponível em: < http://jbehave.org/>. Acesso em: 16 jun. 2011.
24. JUnit web site; Disponível em: <http://www.junit.org/>. Acesso em: 02 jun. 2011.
25. Karner, Cem.; Falk, J.; Nguyen, HQ. **Testing Computer Software**. Second Edition, Willey, 1999.
26. Kaner, Cem.; **An Introduction to Scenario Testing**. http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf. Acessado em 03/07/2011
27. Labiche, Y. and Briand, L. C. **A UML-based approach to system testing**. Software and System Modeling, 1(1):10–42, 2002.
28. Larman, C. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process**, 2nd Edition. Prentice-Hall, 2002.
29. Mayers, G. et al. **The Art of Software Testing**. Wiley. ISBN 0471469122. 2004.
30. Nebut, C.; Fleurey, F.; Jézéquel, J-M.; Traon, Y.L.; **Automatic Test Generation: A Use Case Driven Approach**. IEEE Transactions on Software Engineering 32(3); Los Alamitos, CA: IEEE Computer Society; 2006; pags 140-155.
31. NORTH, D.; **Introducing BDD**. Setembro, 2006. http://dannorth.net/introducing-bdd Acesso em: 26 abr.2011.
32. Proceedings of the 2008 ACM symposium on Applied computing. Fortaleza, Ceara, Brazil, ACM.
33. Ruder, A. **UML-based Test Generation and Execution**. Rückblick Meet- ing. Berlin, 2004.
34. Selenium web site; Disponível em: <http://seleniumhq.org/>. Acesso em: 01 jan. 2011.
35. Somé, S. S. and X. Cheng. **An approach for supporting system-level test scenarios generation from textual use cases**, 2008.
36. SOMMERVILLE, Ian. **Engenharia de Software**, 6a Edição, Addison Wesley, 2003.
37. STAA, A., **Disciplina INF1413 Teste de Software, Notas de Aula, Módulo 04 Especificação**; Disponível em http://www.inf.puc-rio.br/~inf1431/. Acesso em: 16 jun. 2010.
38. Tian, J. Software **Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement**, 2005.
39. UML web site; Disponível em: <http://www.uml.org/>. Acesso em: 20 jun. 2009.
40. Williams, C. E. **Towards a Test-Ready Meta-model for Use Cases**. Workshop of the pUML-Group held together with the UML. 2001, on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists, GI, 2001.
41. Wood, D.; Reis, J.; **Use Case Derived Test Cases.** Harris Corporation.
42. Zambelich, Keith. **Totally Data-Driven Automated Testing**. Disponível em: http://www.sqa-test.com/w\_paper1.html. Acessado em: 10 mai. 2011.

Manual de utilização da ferramenta.

1. **Manutenção de projetos**

O primeiro passo para utilizar a ferramenta cadastrar o projeto. Estes são responsáveis pelo agrupamento dos casos de uso. A manutenção de projetos pode ser dividida nas seguintes funcionalidades: listagem de projetos, busca de projetos, cadastro de projetos e alteração de dados do projetos, conforme descrito abaixo.

* + **Listagem de projetos**

São exibidos os projetos cadastrados na ferramenta, listados em ordem alfabética. A figura abaixo apresenta um exemplo de uma listagem de projetos com limite de três projetos por página.

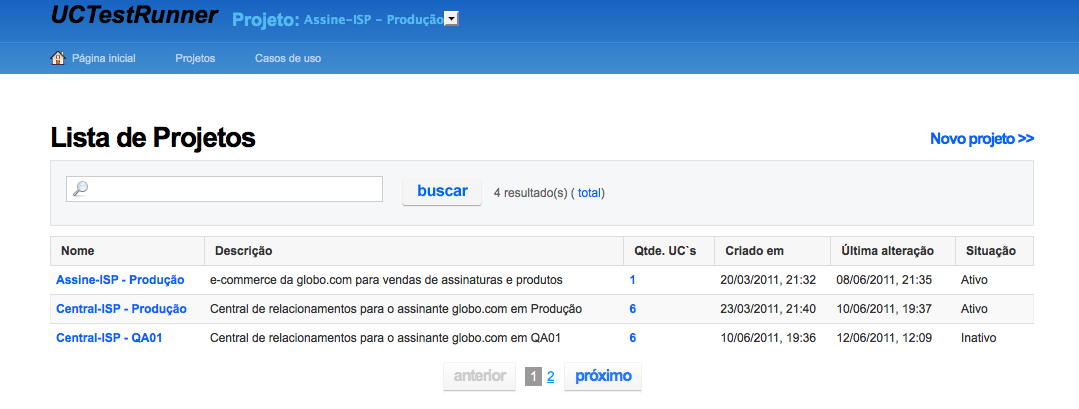


Figura AA 1 - Tela de listagem de projetos cadastrados.

Cada item da lista contém as seguintes informações sobre cada projeto:

**Nome**: nome do projeto. Também utilizado como *hiperlink* para acessar o formulário de alteração dos dados do projeto.

**Descrição**: Descritivo mais detalhado sobre o projeto.

**Qtde de UC’s**: somatório da quantidade de casos de uso cadastrados no projeto. Também um *hiperlink* para acessar a listagem de casos de uso do projeto.

**Criado em**: data de cadastro do projeto na ferramenta.

**Última alteração**: data da última alteração realizada nos dados de cadastro do projeto.

**Situação**: situação do projeto relacionada a execução dos testes. Projeto com situação “Ativo” será utilizado no processo de geração e execução automática dos scripts de testes, o projeto com situação “Inativo” não será utilizado.

* + **Pesquisar** **por projetos**

A pesquisa de projetos foi desenvolvida com intuito de facilitar o acesso a um ou mais projetos, utilizando o “nome do projeto” como chave para realizar o filtro. Para pesquisar um projeto basta informar no campo de pesquisa o “nome do projeto” ou parte dele e acionar o botão pesquisar.

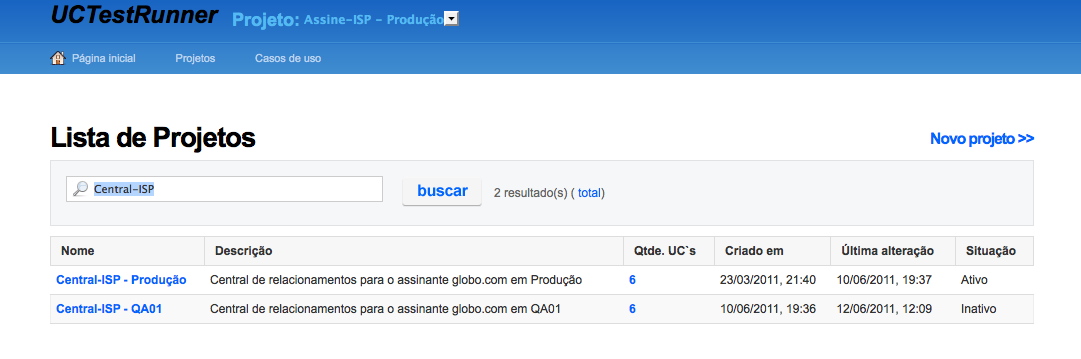


Figura AA 2 - Tela de listagem de projetos filtrados pelo nome.

Conforme exemplo da figura acima, os projetos que contém no nome o texto digitado no campo de pesquisa são exibidos seguindo as regras da listagem de projetos. Neste caso, foram encontrados dois projetos de um total de quatro projetos cadastrados.

* + **Cadastrar projeto**

Para cadastrar um projeto na ferramenta devem ser informados: o nome, a descrição e se o projeto estará disponível para ser utilizado no processo de geração e execução automática de scripts de teste. Ao salvar, o projeto será cadastrado está apto para receber o cadastro de casos de uso. A figura abaixo apresenta o formulário utilizado para cadastrar projetos.

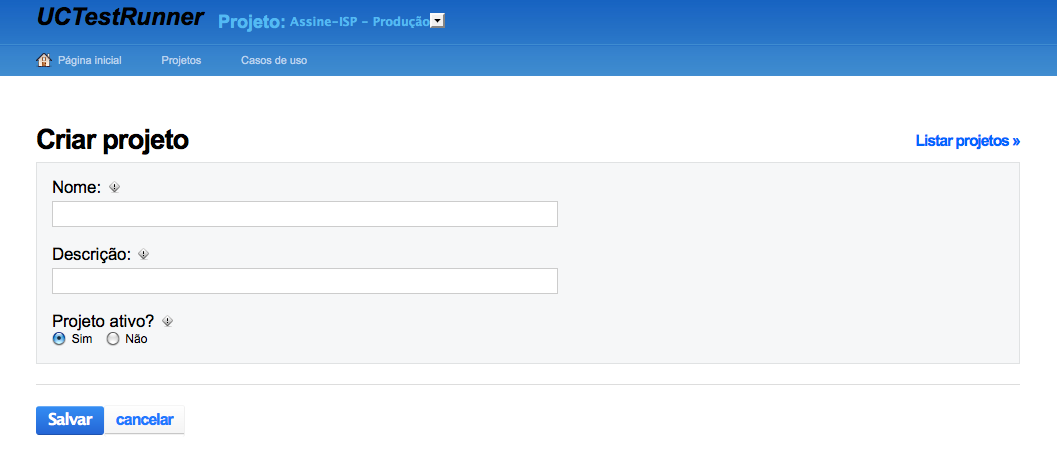


Figura AA 3 - Tela para inclusão de projetos.

* + **Alterar projeto**

Para alterar os dados de um projeto já cadastrado, este deve ser localizado através na listagem de projetos, selecionar o *hiperlink* no nome do projeto e no formulário alterar os dados e salvar a operação. Conforme ilustrado na figura abaixo na tela de alteração de dados do projeto.

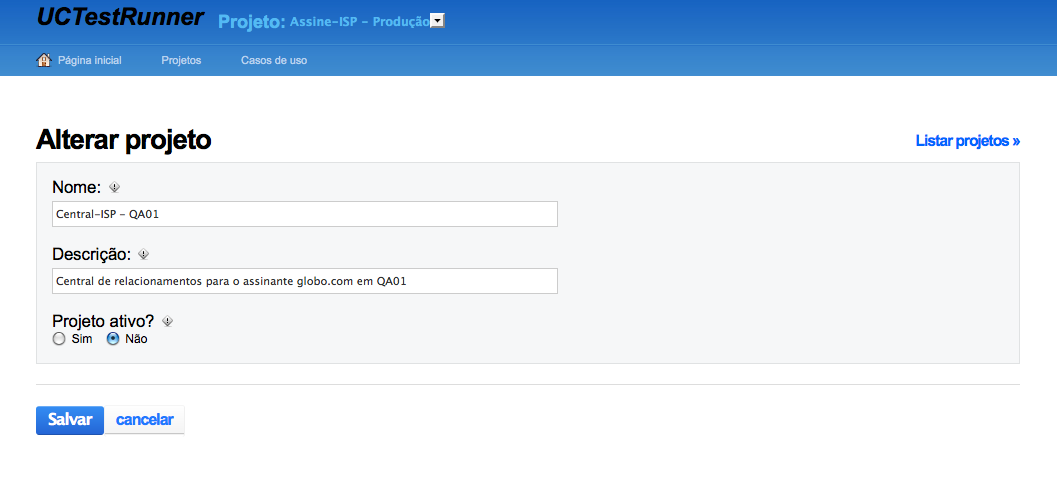


Figura AA 4 - Tela para alterar dados de projetos.

1. **Casos de uso**

Após realizar o cadastro de um ou mais projetos, podemos iniciar o cadastro dos casos de uso. As funcionalidades para cadastrar e manter o cadastro dos casos de uso estão divididas em: listagem de casos de uso, busca por casos de uso, cadastro de caso de uso e alteração de informações do caso de uso, conforme descritos abaixo.

* + **Listar casos de uso**

A lista de casos de uso exibe os casos de uso cadastrados para o projeto selecionado na lista de opções no menu fixo superior da ferramenta. Os casos de uso são listados em ordem alfabética, com algumas informações do caso de uso e com limite configurável da quantidade de casos de uso por página. O figura abaixo apresenta um exemplo da listagem de todos os casos de uso cadastrados para o projeto “Central-ISP – QA01”.

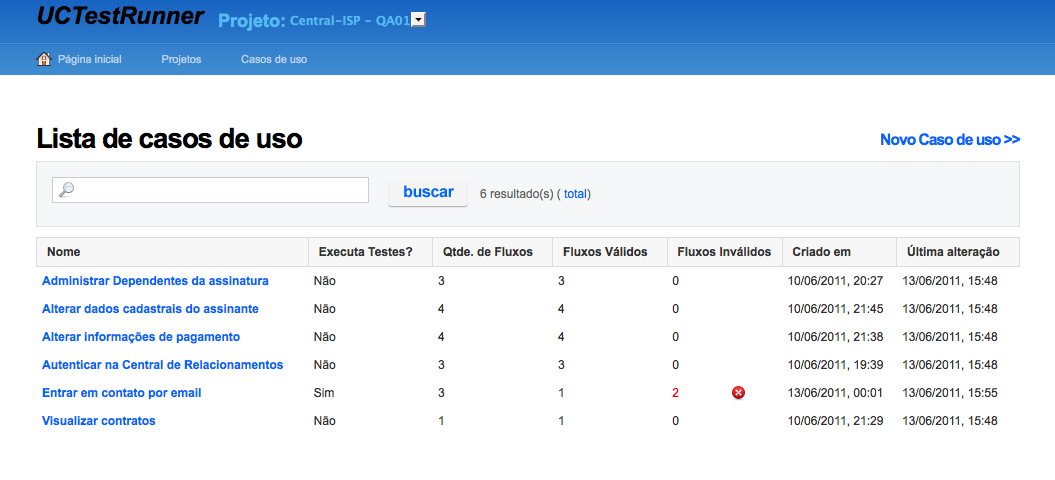


Figura AA 5 - Lista de casos de uso cadastrados para um projeto.

Cada item da lista contém as seguintes informações sobre cada caso de uso:

**Nome**: nome do caso de uso. Também utilizado como *hiperlink* para acessar o formulário de alteração do caso de uso.

**Executa Testes?**: propriedade do caso de uso utilizada na geração e execução automática do testes. Caso a propriedade seja “Sim“, o caso de uso será utilizado no processo de geração e execução automática dos testes, caso contrário não será utilizado no processo.

**Qtde de Fluxos**: somatório da quantidade de fluxos do caso de uso.

**Fluxos Válidos**: quantidade de fluxos cujos testes gerados e executados automaticamente estão em conformidade com o comportamento cadastrado.

**Fluxos Inválidos**: quantidade de fluxos cujos testes gerados e executados automaticamente não estão em conformidade com o comportamento cadastrado. Neste caso, se existir algum fluxo inválido, é apresentado um alerta neste campo indicando que houve um problema no processo. Conforme ilustrado no exemplo da figura anterior.

**Criado em**: data de cadastro do caso de uso.

**Última alteração**: data da última alteração realizada nos dados de cadastro do caso de uso.

* + **Pesquisar casos de uso**

A funcionalidade para pesquisar casos de uso foi desenvolvida com intuito de facilitar o acesso direto a um ou mais casos de uso, utilizando além do nome do caso de uso como parâmetro de pesquisa, o projeto selecionado no menu superior da ferramenta.

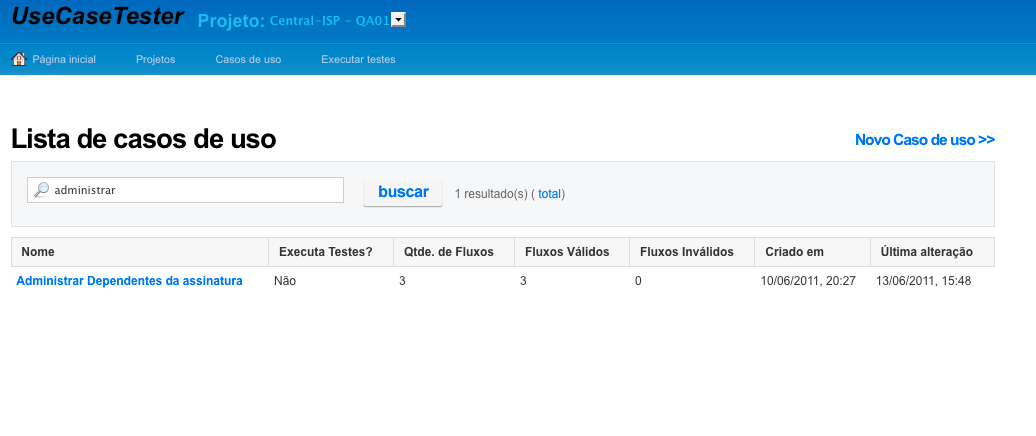


Figura AA 6 - Lista de casos de uso cadastrados para um projeto.

Para realizar uma pesquisa por um ou mais casos de uso, basta selecionar o projeto no menu superior da ferramenta e informar dados do nome do caso de uso no campo do formulário de pesquisa e solicitar a busca. A figura acima apresenta o exemplo de uma pesquisa por casos de uso do projeto “Central-ISP –QA01” cujo nome contém “administrar”.

* + **Cadastrar caso de uso**

Para cadastrar um caso de uso, o primeiro passo é selecionar no menu superior da ferramenta o projeto para qual o caso de uso fará parte e então informar todos os campos solicitados no formulário. Ao confirmar o cadastro, o caso de uso será cadastrado dentro do projeto selecionado e poderá ser visualizado na listagem de casos de uso desse projeto. A figura abaixo apresenta um exemplo do formulário de cadastro de caso de uso, neste caso, o caso de uso a ser cadastrado será um caso de uso do projeto “Central-ISP – QA01”.

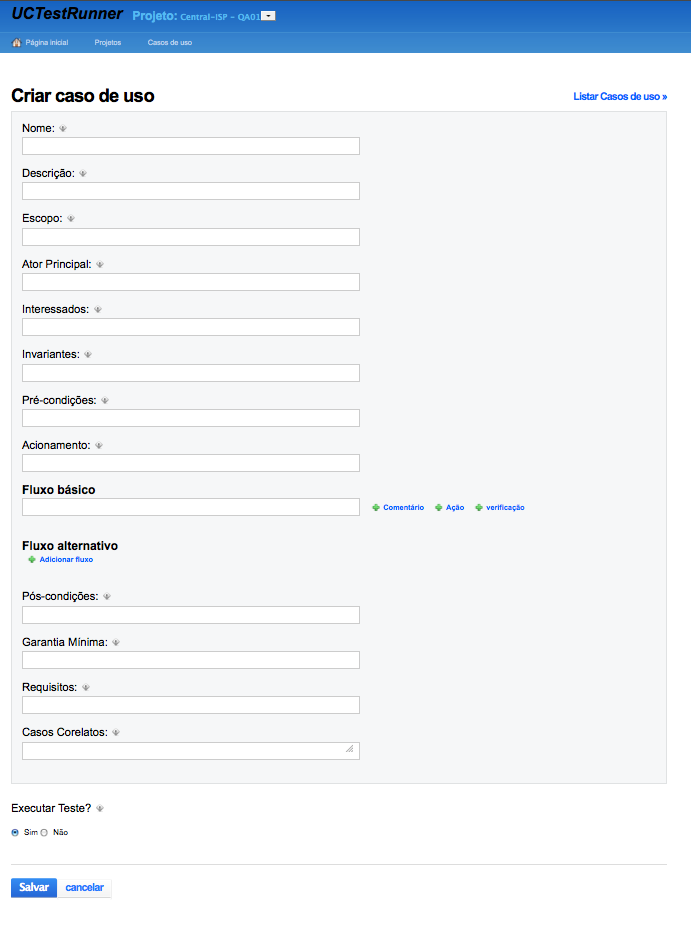


Figura AA 7 - Formulário para cadastro do caso de uso.

Abaixo a descrição dos campos do formulário de cadastro de casos de uso:

**Nome**: nome (identificação) do caso de uso.

**Descrição**: descrição resumida do objetivo principal do caso de uso.

**Escopo**: o que é abrangido pelo caso de uso.

**Ator Principal**: nome (identificação) do ator principal.

**Interessados**: descrição do interesse ou objetivo do ator.

**Invariantes**: condições que deverão estar satisfeitas antes e após a execução do caso de uso.

**Pré-condições**: condições que precisam estar satisfeitas antes de iniciar o caso de uso.

**Acionamento**: como e quando o caso de uso deve ser executado.

**Fluxo principal**: sequência de ações esperadas.

**Fluxos alternativos**: sequências de ações que tratam de exceções e desvios do fluxo principal.

**Pós-condições**: condições que devem estar satisfeitas ao terminar a execução normal do caso de uso.

**Garantia mínima**: condições que devem estar satisfeitas sempre, independente de como termine o caso de uso.

**Requisitos**: requisitos adicionais, tais como requisitos não funcionais e outras características desejáveis.

**Casos de uso correlatos**: Relação de casos de uso correlacionados com o presente caso de uso.

* + **Manutenção de fluxos no caso de uso**

Para cadastrar os fluxos de eventos do caso de uso, o formulário de cadastro disponibiliza como obrigatório o fluxo principal e valida se o mesmo foi informado antes de realizar o cadastro. Os fluxos alternativos não são obrigatórios e podem ser incluídos dinamicamente ao formulário, bem como podem ser excluídos ou até mesmo reordenados através do recurso *drag-and-drop* (arrastar e largar) implementado no formulário.

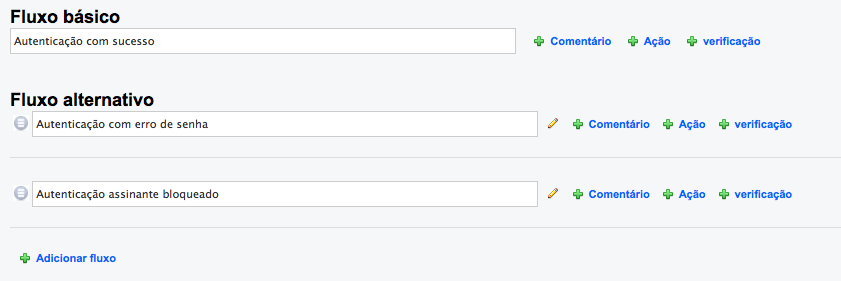


Figura AA 8 - Inclusão dos fluxos.

No exemplo da figura acima o caso de uso tem o fluxo principal obrigatório e dois fluxos alternativos inseridos dinamicamente no formulário. Note no exemplo que apenas os fluxos alternativos disponibilizam a opção para reordenação e a opção de exclusão. Não há limites para o número de fluxos alternativos que podem ser cadastrado em cada caso de uso.

* + **Manutenção dos passos do fluxo**

Os fluxos de eventos podem receber o cadastro de três tipos de passos para descrever seu comportamento, são eles: comentário, ação e verificação.

**Comentário** – o passo comentário, conforme ilustrado abaixo na figura abaixo, é um campo texto livre para digitação e pode ser utilizado para enriquecer a documentação do fluxo. Passos desse tipo não tem utilidade no processo de geração dos testes, apenas complementam a documentação.



Figura AA 9 - Exemplo de um passo do tipo comentário.

**Ação** – cada passo do tipo ação cadastrado no fluxo de eventos será responsável por realizar uma interação com o formulário web através da integração com o framework de testes *Selenium*. A figura abaixo ilustra a inclusão desse tipo de passo dentro do fluxo e também exibe a lista de opções com os tipos de ações disponíveis.

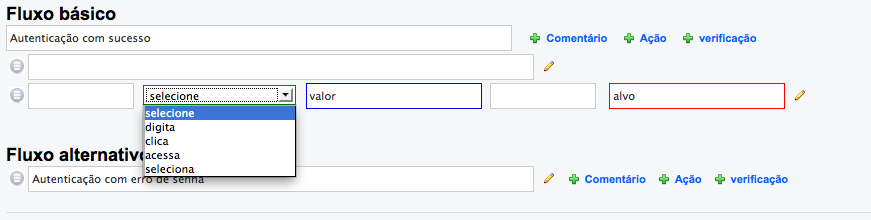


Figura AA 10 - Inclusão do passo do tipo ação.

A configuração dos campos do passo do tipo ação pode ser alterada dependendo do tipo de ação escolhida para o passo na lista de seleção. O exemplo da figura 11 abaixo apresenta a configuração do passo para cada tipo de ação disponível na lista de seleção apresentada na figura 10.

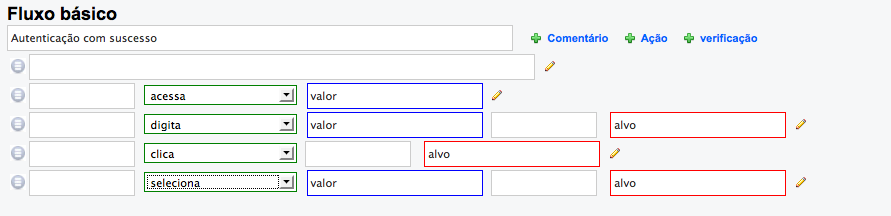


Figura AA 11 - Diferentes opções de ação.

**Verificação** – esse tipo de passo é responsável por verificar, através da integração com o framework de testes *Selenium*, se o texto cadastrado no passo existe no formulário web da aplicação. Isso possibilita que a ferramenta verifique as mensagens apresentadas pela aplicação web e consiga verificar se o fluxo atingiu o objetivo esperado. Na figura abaixo um exemplo da inclusão desse tipo de passo em um fluxo.



Figura AA 12 - Inclusão do passo do tipo verificação.

Para os três tipos de passos apresentados (documentação, ação e verificação), assim como na manutenção de fluxos, os passos incluídos no fluxo podem ser excluídos e também reordenados dentro do próprio fluxo através do recurso *drag-and-drop*.

* + **Alterar caso de uso**

Para alterar os dados de um caso de uso já cadastrado, é necessário localizar o caso de uso na listagem de casos de uso, selecionar o hiperlink no nome do caso de uso e no formulário de casos de uso alterar os dados básicos e para alterar os fluxos e os passos, proceder conforme descrito nos itens descritos acima: “manutenção de fluxos” e “manutenção dos passos do fluxo”.

1. **Geração e execução dos scripts de teste**

O processo de geração e execução dos scripts de testes também foi desenvolvido na ferramenta “*UseCaseTester”* e pode ser acionado espontaneamente através da opção “Executar testes” no menu superior da ferramenta, conforme figura abaixo ou são executados automaticamente em períodos pré-determinados na funcionalidade de agendamento. Atualmente este parâmetro só pode ser alterado no código fonte da ferramenta e foi configurado para executar os testes em períodos de três horas, ou seja, em vinte e quatro horas os testes serão executados automaticamente oito vezes.



Figura AA 13 - Opção “Executar Testes” no menu superior.

Quando a solicitação de execução de testes é disparada, tanto pela forma espontânea quanto pelo agendamento automático, a ferramenta inicia o processo de geração e execução dos testes executando basicamente os seguintes atividades:

**Buscar projetos** - busca os projetos cadastrados com atributo “Executar Testes? = Sim”;

**Buscar casos de uso e iniciar janela do navegador** - para cada projeto com atributo “Executar Testes? = Sim”, busca os casos de uso que também tem o atributo “Executar Testes? = Sim” e utiliza a integração com o *Selenium* para abrir uma nova janela do navegador web;

**Buscar fluxos, iniciar e fechar janela do navegador** - para cada caso de uso encontrado no item anterior, busca todos os fluxos de eventos do caso de uso. Através da integração com *Selenium*, fecha as sessões da janela do browser e abre uma nova sessão para garantir que cada fluxo seja executado utilizando uma nova sessão da janela do navegador web, evitando interferências das sessões antigas que podem distorcer o resultado da navegação;

**Buscar, ordenar e negativar passos do fluxo** - para cada fluxo, busca e ordena todos os passos e atualiza no banco de dados o atributo “status\_test” de cada passo para “N”, negativando o passo. Isso porque cada passo executado na próxima atividade, vai atualizar este atributo para “S” se o passo for executado com sucesso;

**Gerar e executar o script de teste** - Cada passo é identificado e é gerado um script *Selenium* com as informações desse passo, para então, esse script executar uma interação com o formulário web. Se o script for executado corretamente o atributo “status\_test” do passo é atualizado no banco de dados com “S”, indicando sucesso, caso contrário, o atributo é atualizado para “N” indicando que o passo não foi executado com sucesso. Quando um passo do fluxo não for executado corretamente, o restante dos passos desse fluxo não serão executados e o atributo “status\_test” desse passos não executados ficarão com o valor “N”, atualizado no atividade “Buscar, ordenar e negativar passos do fluxo**”**.

Para facilitar o entendimento da geração e execução dos scripts de teste descritos nas atividades acima, a figura abaixo apresenta um resumo adaptado e comentado do código utilizado na ferramenta. A linguagem de programação utilizada no desenvolvimento é o *Ruby on Rails*.

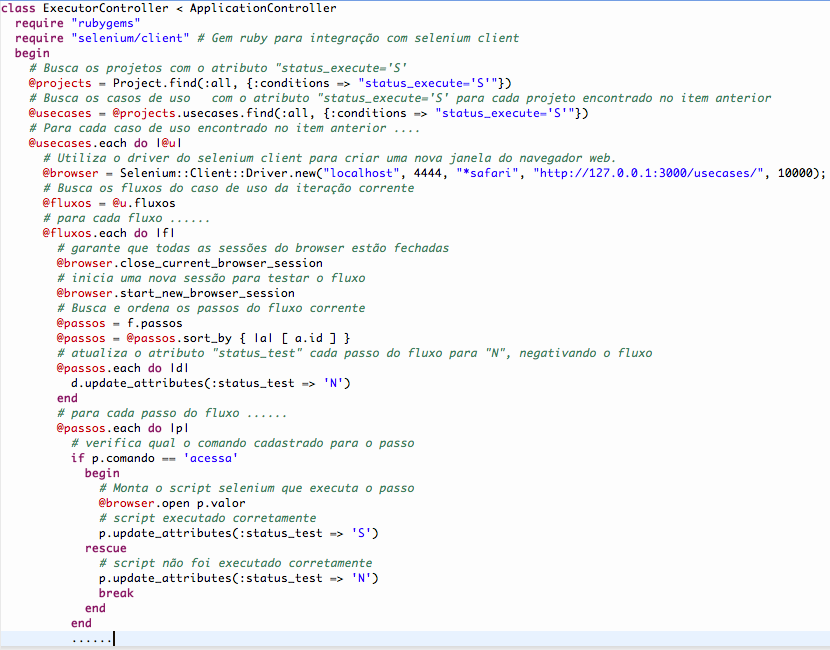


Figura AA 14 - Trecho do código fonte que executa as atividades de gerar e executar os scripts de teste.

1. **Resultado da execução dos testes**

O resultado da geração e execução automática dos testes pode ser analisado através de três diferentes visões: visão geral, visão do projeto e visão do caso de uso. Na visão geral, localizada na página inicial da ferramenta, é possível verificar rapidamente, através da lista de projetos cadastrados, dados da última execução automática dos testes para cada projeto da lista. Esses dados são atualizados automaticamente a cada minuto e exibem as seguintes informações:

**Projeto**: nome do projeto.

**Caso(s) de uso**: quantidade de casos de uso do projeto.

**Qtde. de fluxos**: quantidade de fluxos do projeto.

**Fluxos válidos**: quantidade de fluxos do projeto que executaram os testes corretamente.

**Fluxos Inválidos**: fluxos cujos testes não executaram corretamente.

**Última execução**: data da última execução do dos testes.

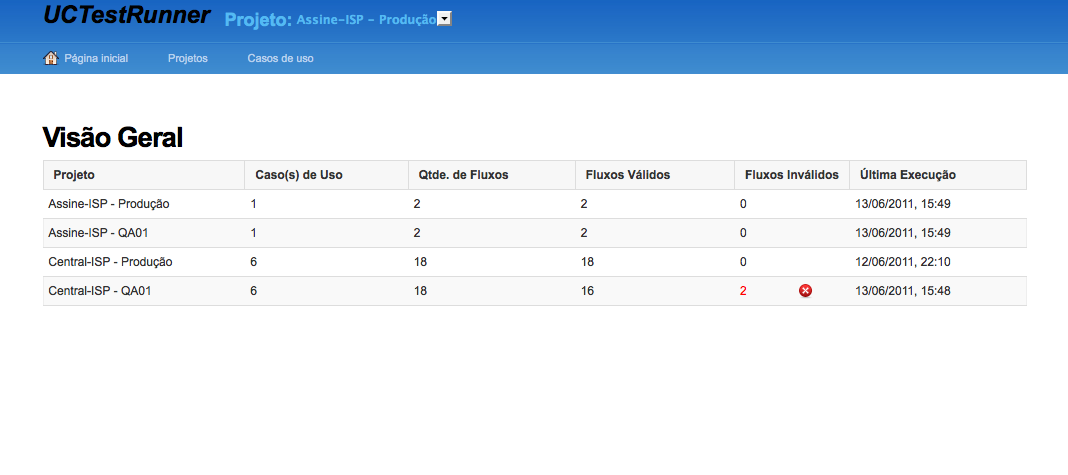


Figura AA 15 - Visão do resultado dos testes para todos os projetos.

Conforme apresentado no exemplo da figura acima, na última execução dos testes a ferramenta indicou falha no comportamento de dois fluxos de um projeto.

Na visão do projeto, localizada na lista de casos de uso do projeto é possível localizar os casos de uso que apresentaram falha. Na figura abaixo os dois fluxos de um caso de uso estão indicados como inválidos.

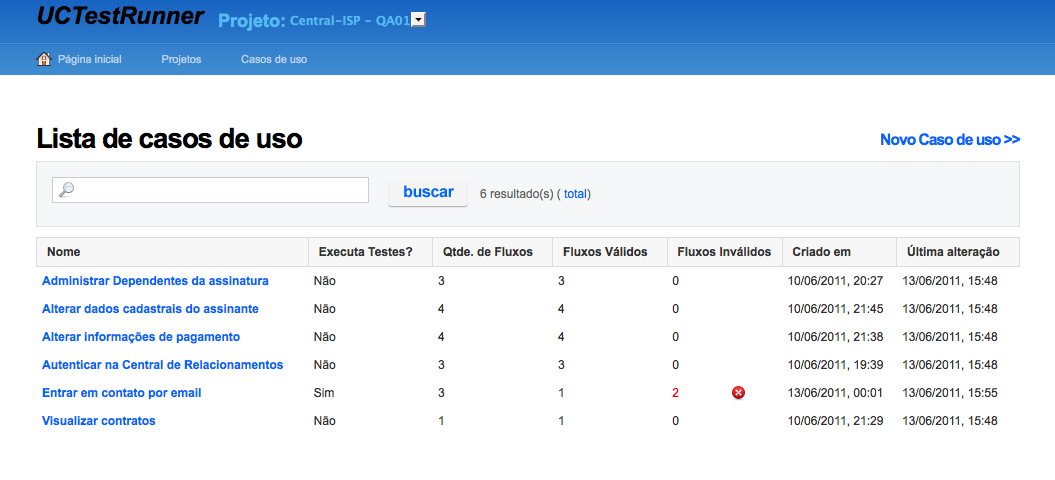


Figura AA 16 - Visão do resultado dos testes por casos de uso de um projeto.

Na visão do caso de uso, localizada no formulário de cadastro e manutenção de casos de uso, é possível identificar os fluxos e o passo de cada fluxo com comportamento indesejado, conforme exemplo da figura abaixo.



Figura AA 17 - Visão do resultado dos testes por passos do fluxo.

Ou seja, para verificar qual passo falhou na geração e execução automática dos testes, é preciso identificar o projeto na página inicial da ferramenta, acessar a lista de casos de uso desse projeto e abrir o formulário do caso de uso.