UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO E REUTILIZAÇÃO DE TESTES AUTOMATIZADOS EM APLICAÇÕES WEB

TRABALHO DE GRADUAÇÃO

Lucas Antunes Amaral

Santa Maria, RS, Brasil

2014

DESENVOLVIMENTO E REUTILIZAÇÃO DE TESTES AUTOMATIZADOS EM APLICAÇÕES WEB

Lucas Antunes Amaral

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientadora: Profª. Drª. Andrea Schwertner Charão

Santa Maria, RS, Brasil

2014

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO E REUTILIZAÇÃO DE TESTES AUTOMATIZADOS EM APLICAÇÕES WEB

elaborado por
Lucas Antunes Amaral

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Andrea Schwertner Charão, Drª.
(Presidente/Orientadora)

Giliane Bernardi, Profª. Dr. (UFSM)

Adriano Pereira, (UFSM)

Santa Maria, de Outubro de 2014.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

DESENVOLVIMENTO E REUTILIZAÇÃO DE TESTES AUTOMATIZADOS EM APLICAÇÕES WEB

AUTOR: LUCAS ANTUNES AMARAL
ORIENTADORA: ANDREA SCHWERTNER CHARÃO
Local da Defesa e Data: Santa Maria, de Outubro de 2014.

A constante busca pela qualidade de uma solução em forma de software, fez com que empresas do ramo de desenvolvimento, aderissem e enxergassem a importância da realização de testes automatizados em seus sistemas. A partir deste cenário, surgiram inúmeras ferramentas e *frameworks* para suprir esta demanda, que se propõe a ampliar a otimização de tempo e eficácia das aplicações implementadas, visando uma garantia maior na qualidade das mesmas. Contudo, é sabido que criar um novo teste para cada nova funcionalidade ou demanda do sistema, se torna muito custoso, sendo necessário um grande desprendimento de recursos humanos. Assim, este trabalho objetiva apresentar *scripts* de testes automatizados para sistemas web, que possam, de maneira mais genérica e com poucas alterações, serem reutilizados, de forma escalável, para novos casos de testes, que sigam o mesmo escopo.

Palavras-chave: Qualidade de Software. Testes automatizados de Software. Linguagens de Programação. Selenium HQ. Cucumber.

SUMÁRIO

1 INTRODUÇÃO

Dada a atual conjectura do mercado de desenvolvimento de software, é fundamental para que uma aplicação se mantenha viva de forma competitiva, apresentar diferenciais ao seu público alvo. Desta maneira a área de qualidade de software ganha cada vez mais espaço dentro das empresas de TI e, em especial, as ferramentas e metodologias de teste de software ganham maior visibilidade.

Os testes de software podem ocorrer em todas as etapas do desenvolvimento e de diferentes formas, contudo, sempre objetivam atender na totalidade os requisitos do sistema e, simultaneamente, amplificar a qualidade da solução codificada. São inúmeras as vantagens de se utilizar testes automatizados ao invés dos testes manuais em uma aplicação. Apesar de, aparentemente, ser mais prático e rápido realizar um teste manual, a cada nova alteração em um módulo do sistema, o teste tem que ser todo refeito e a tendência que novos erros sejam gerados até mesmo em funcionalidades já testadas é enorme, problema este que não ocorre quando a abordagem escolhida é a automatização dos testes.

Mesmo apresentando grandes vantagens, os testes automatizados demandam um grande custo inicial em sua codificação e, com isso, aumenta o envolvimento da equipe de qualidade. Pensando nesse problema, podemos buscar formas alternativas para que se possa usufruir de todas estas virtudes dos testes automatizados, e, ao mesmo tempo, utilizar de forma eficiente os recursos disponíveis em uma instituição.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo principal apresentar um conjunto de casos de teste e testes que possam ser reutilizados de forma otimizada em novas funcionalidades de uma aplicação ou em sistemas que sigam os mesmos padrões e comportamento dos softwares conhecidos.

1.1.2 Objetivos Específicos

- Apontar diferenças de casos de testes;
- Gerar testes automatizados reaplicáveis em novos casos;

- Gerar casos de testes genéricos para um escopo definido.

1.2 Justificativa

A qualidade de software é uma das variáveis essenciais para que um projeto de software tenha sucesso. Sendo assim, torna-se cada vez mais necessário a inserção de testes automatizados em projetos web, agregando aos mesmos uma maior confiabilidade e redução nos possíveis erros que o sistema possa apresentar. Para que haja a possibilidade de aumentar a qualidade dos sistemas, sem que seja necessário uma maior demanda de recursos humanos para a área de qualidade, podemos adotar práticas de reuso de códigos de testes, visando maximizar a produtividade e eficiência, além de, simultaneamente, obter um produto final com uma garantia de qualidade superior.

2 FUNDAMENTOS E REVISÃO DE LITERATURA

Neste capítulo, serão apresentados conceitos relativos os conteúdos abordados neste trabalho, descrevendo, qualidade de software, ferramentas de teste de software, assim como o reuso de testes.

2.1 Qualidade de Software

A Qualidade de software é uma subárea oriunda da engenharia de software, que tem com foco central apresentar metodologias, procedimentos e métricas que garantam a qualidade no processo de desenvolvimento de um sistema. Apesar de ocorrer no processo, a Qualidade de software objetiva obter qualidade no produto final, e, com isso, consiga contemplar na totalidade os requisitos tratados com o cliente ao longo do processo. VASCONCELOS et al. (VASCONCELOS et al., 2006) afirma que a qualidade de software está diretamente relacionada a um gerenciamento rigoroso de requisitos, uma gerência efetiva de projetos e em um processo de desenvolvimento bem definido, gerenciado e em melhoria contínua. Afirma também, que atividades de verificação e uso de métricas para controle de projetos e processo também estão inseridas nesse contexto, contribuindo para tomadas de decisão e para antecipação de problemas.

Mas como medir a qualidade de um sistema em questão? Para responder este questionamento, Garvin (GARVIN, 1987) propõe o conceito que ele chama de oito dimensões, que seriam em ordem, qualidade do desempenho, qualidade dos recursos, confiabilidade, conformidade, durabilidade, facilidade de manutenção, estética e percepção. Segundo GARVIN, atendendo a estes oito critérios, o sistema apresentará qualidade. PRESSMAN (PRESSMAN, 2011) complementa a definição de Garvin mesclando-a com a norma ISO 9126, e aponta os fatores críticos para o sucesso neste caso, como sendo: Intuição, Eficiência, Robustez e Riqueza.

2.1.1 Qualidade do processo

A qualidade de software é largamente determinada pela qualidade dos processos utilizados para o desenvolvimento. Deste modo, a melhoria da qualidade de software é obtida pela melhoria da qualidade dos processos (KOSCIANSKI; SOARES, 2007).

2.1.2 Qualidade do produto

Existe uma relação direta entre qualidade de produto e qualidade do processo, pois, para obtenção da qualidade do produto final, faz-se necessário adquirir primeiramente qualidade nos processos que compõem o desenvolvimento do mesmo. Avaliar a qualidade de um produto de software é verificar, através de técnicas e atividades operacionais o quanto os requisitos são atendidos. Tais requisitos, de uma maneira geral são a expressão das necessidades, explicitados em termos quantitativos ou qualitativos, e têm por objetivo definir as características de um software, a fim de permitir o exame de seu atendimento (KOSCIANSKI; SOARES, 2007).

2.1.3 Testes de Software

É a atividade responsável por apresentar os erros existentes em um determinado programa, por isso, pode ser vista como uma atividade destrutível, pois visa expor os defeitos para depois corrigir os mesmos, e, de preferência, em um estágio inicial. Quanto mais tarde um defeito for identificado mais caro fica para corrigi-lo e mais, os custos de descobrir e corrigir o defeito no software aumentam exponencialmente na proporção em que o trabalho evolui através das fases do projeto de desenvolvimento (BOEHM; BROWN; LIPOW, 1976). O teste possibilita também, validar se os requisitos inicias do sistema, alinhados pelos *stakeholders*, estão contemplados em sua plenitude.

Apesar de não ser possível, através de testes, provar que um programa está correto, os testes, se conduzidos sistemática e criteriosamente, contribuem para aumentar a confiança de que o software desempenha as funções especificadas e evidenciar algumas características mínimas do ponto de vista da qualidade do produto (MALDONADO et al., 2004). Sendo assim, se faz essencial o mapeamento de um processo de testes para que se possa criar garantias e métricas que reduzam os erros, maximizando a qualidade, CRESPO et al. (CRESPO et al., 2004) descreve o processo de teste, como sendo a composição de quatro macro etapas: Planejamento, projeto, execução e acompanhamento dos testes de unidade.

2.1.4 Estratégias de Testes

A Estratégia de testes se caracteriza pela definição da abordagem geral a ser aplicada nos testes, descrevendo como o software será testado, identificando os níveis de testes que serão aplicados, os métodos, técnicas e ferramentas a serem utilizada (RIOS, 2006). Existem muitas

estratégias mas podemos destacar entre elas: Teste de unidade, integração, sistema, aceitação e regressão, funcional e carga. Se pode também, mesclar mais de uma estratégia com o intuito de reduzir os possíveis defeitos que o sistema venha a ter.

2.2 Ferramentas para teste de software

Existem muitas ferramentas desenvolvidas para realização de testes de software web, neste trabalho serão descritas três(3) dentre elas, que serão as ferramentas utilizadas no desenvolvimento dos testes estudados.

2.2.1 Cucumber

Cucumber é um ferramenta de desenvolvimento de testes, voltado para sistemas web, que adota uma linguagem de alto nível bem próxima à uma linguagem natural e tem suas origens fixadas sobre a metodologia BDD (Behavior Driven Development). Cucumber é escrita em linguagem Ruby, mas pode ser utilizada para executar especificações de aplicações escritas em qualquer linguagem (NUNES, 2009).

A escolha dessa ferramenta, baseou-se na fácil transcrição dos requisitos do sistema para a linguagem em questão, tornando possível conferir se os requisitos estão contemplados pelas funções e métodos descritos no sistema *web*. LOPES(LOPES, 2011) compara Cucumber com o *software* Capybara e afirma que o primeiro apresentar um código mais legível e amigável. Uma observação é que o código com Capybara faz referência para vários detalhes de implementação, enquanto o código do Cucumber reserva isso apenas para os *Steps* e não para o arquivo de feature (LOPES, 2011).

A ferramenta funciona basicamente através da leitura de arquivos com a extensão feature, os quais descrevem em linguagem natural uma funcionalidade e casos de teste, conhecidos como cenários. Como os testes estão escritos em uma linguagem natural, e não de programação, Cucumber precisa pesquisar pelo código associado aos passos que formam o cenário em arquivos auxiliares (SCHMITZ; BECKER; BERLATTO, 2013). Cucumber executa seus arquivos *.feature*, e esses arquivos contêm especificações executáveis escritos em uma linguagem chamada Gherkin (REFERENCE CUCUMBER.IO, 2015), que, possui um layout bem definido. Inicia pela descrição de uma funcionalidade, que por sua vez possui cenários, onde, um cenário é descrito da seguinte forma: *Dado* alguma condição *Quando* outra condição *E* terceira

condição *Então* faça algo, conforme ilustra a figura 2.1.

```
Feature: Sign up

  Scenario: Successful sign up
    Given I am on the homepage
    When I follow the sign up link
    And I fill out the form with valid details
    Then I should receive a confirmation email
    And I should see a personalized greeting message
```

Figura 2.1 – Um exemplo de código cucumber.

2.2.2 Selenium HQ

É um framework open source, utilizado para automatização de testes funcionais em aplicações web (CHIAVEGATTO[1] et al., 2013). Segundo PEREIRA (PEREIRA, 2012), se trata de uma ferramenta de fácil uso e eficiente para desenvolver casos de teste, permitindo os testes de aceitação ou funcional, regressão e de desempenho. Selenium trabalha como um plugin do navegador Firefox, o mesmo traz muita praticidade pois permite que se possa capturar cliques e valores digitados transformando-os em um caso de teste. Ele é composto por quatro ferramentas: Selenium IDE, Selenium Grid, Selenium RC e Selenium WebDrive.

A utilização de comandos no Selenium consiste em digitar o comando seguido de dois parâmetros tal como por exemplo *verifyText //div//a[2] Login*. Dependendo do comando, os parâmetros poderão ser opcionais, aliás, alguns comandos não necessitam de parâmetros para serem executados (SIXPENCE; ADÃO; SMITH, 2011).

Selenium foi escolhida como umas das ferramentas no desenvolvimento deste projeto pelo fato de ser um *framework open source* que possui um vasto leque de ferramentas para testes de sistemas web. Outro fator importante para sua escolha, é a possibilidade de interação do Selenium HQ com o Cucumber. Ela se destaca entre as demais ferramentas gratuitas pelo fato de ser a mais completa, permitindo integração com várias linguagens, outros *frameworks*, além de suportar inúmeros navegadores e sistemas operacionais (PEREIRA, 2012).

2.2.3 JUnit

JUnit é uma ferramenta de apoio ao teste unitário, a qual auxilia desenvolvedores na automação dos testes e verificação dos resultados (BIASI, 2006). A escolha desta ferramenta,

como parte integrante da solução desenvolvida deveu-se ao fato de a mesma ser voltada para sistemas desenvolvidos na linguagem de programção *Java*, além de ser altamente versátil, possibilitando assim, integração em um único código da mesma com os demais *frameworks* que serão utilizados(*Selenium* e *Cucumber*).

Se justifica a sua utilização neste trabalho, por tratar-se de uma *API* que viabiliza a comparação de um valor obtido em algum teste com o valor esperado pelo mesmo. Desta forma, pode-se validar e verificar se a funcionalidade de um sistema está funcionando adequadamente.

2.3 Reuso de testes

Visando melhor aproveitar os recursos existentes em uma instituição e ainda assim apresentar garantias na qualidade do produto/serviço entregues aos clientes, desenvolvedores do mundo todo, começaram a apresentar teses e modelos que buscam criar testes genéricos e padronizados. Um padrão é um pedaço de informação instrutiva e nomeada, que captura a estrutura essencial e *"insights"* de uma família bem sucedida de soluções aprovadas para um determinado problema, o qual surge em um determinado contexto (CAGNIN et al., 2004).

GUIZZARDI diz que por razões histórias a área de desenvolvimento de software não atingiu a maturidade que outras áreas da engenharia atingiram, complementa afirmando, que apesar disso, é inegável que algum avanço tenha sido alcançado, pois a forma de realização dessa atividade evoluiu de uma atividade realizada de forma quase artesanal, para um processo de desenvolvimento bem estruturado e que, nos melhores casos, contempla inclusive atividades de gerência e avaliação da qualidade (GUIZZARDI, 2000). Com isso, a reutilização dos testes já codificados, se mostra uma importante prática no desenvolvimento e que ainda apresenta muitas incógnitas e possibilidades para as equipes de TI, principalmente na geração de casos de testes que possam ser reutilizados em situações que apresentem um padrão parecido com os casos já conhecidos.

3 DESENVOLVIMENTO

Abaixo, serão descritas as atividades desenvolvidas afim de alcançar os objetivos propostos por este trabalho. Será apresentada a estrutura básica dos sistemas escolhidos para inserção dos testes automatizados, assim como trechos dos códigos desenvolvidos e estrutura das soluções encontradas ao longo do desenvolvimento do projeto.

3.1 Delimitação de escopo

Como escopo, foram delimitados alguns pré-requisitos necessários para utilização da solução proposta neste trabalho, onde, o sistema deve ser um software *web*, desenvolvido na linguagem de programação *Java*, podendo apresentar também funções *Ajax* e *Javascript*.

3.2 Visão geral da solução

Em um primeiro momento, com objetivo de automatizar os testes de uma aplicação afim de reduzir o esforço na confecção e codificação de novos testes para as equipes de desenvolvimento e testes de uma empresa, escolheu-se desenvolver uma solução baseada em três etapas que são descritas conforme imagem abaixo.

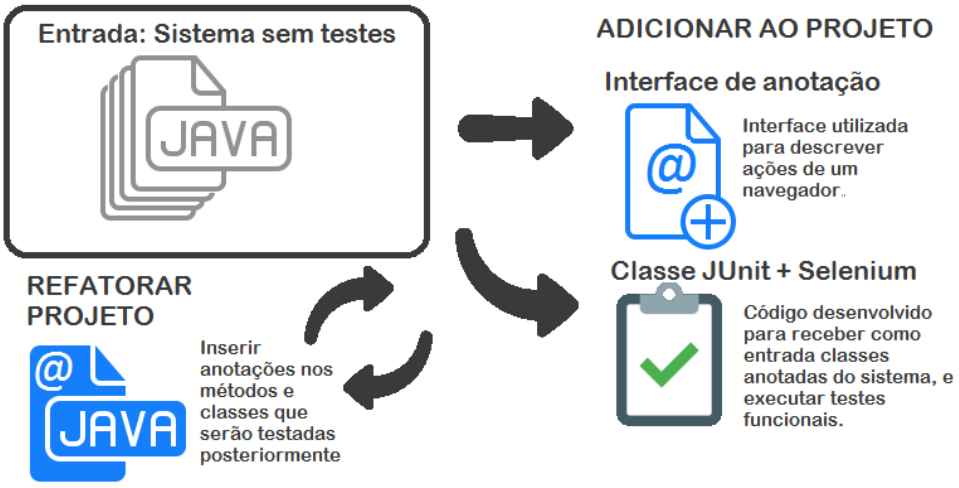

Figura 3.1 – Estrutura da solução.

Temos como ponto de partida um sistema onde se possuí pouco ou nenhum teste codificado. Para o mesmo, descrevemos três fases no desenvolvimento de uma solução em forma de testes para o sistema, que seriam: criar uma classe de testes genérica, responsável por realizar

os testes necessários na aplicação, descrever uma interface de anotação que representaria ações de elementos web (*HTML*) que seriam testados posteriormente e por fim inserir as anotações nos métodos e classes do sistema que devem ser testados nesta aplicação.

3.3 Classe genérica de testes

Por tratar-se de uma abordagem para sistemas-web, optou-se pela utilização de dois *frameworks* de testes que possibilitam interação com elementos *web* para confecção da classe responsável pela execução dos testes automatizados, o primeiro seria o *Selenium*. Através da utilização da classe `WebDrive`, contida no pacote do mesmo, podemos descrever uma sequência de passos que são executados por um navegador, como cliques e inserção de dados em campos de um formulário. Sendo assim instanciando um objeto *WebDrive* podemos iniciar um navegador e realizar uma série de instruções pré-determinadas para validar se a aplicação comporta-se de forma adequada.

Por fim, necessitamos validar se após realizar o preenchimento de formulários do *software*, o resultado obtido condiz com o resultado esperado. Neste momento é onde faz-se necessário a adição de trechos de códigos *Junit* na classe de teste desenvolvida, pois o mesmo disponibiliza em seu pacote, um conjunto de métodos *assert*, com os quais, se faz possível comparar, por exemplo, se após salvar um formulário de cadastro, a mensagem de "cadastrado com sucesso"aparece na tela.

3.4 Interface de anotação

Após finalizar o desenvolvimento da classe de testes genérica, passou-se a existir a necessidade de criar um mecanismo para informar para a classe de testes, qual seriam as validações necessárias. Para esta tarefa, por tratar-se de projetos *Java*, resolveu-se criar uma *interface* de anotação que seria vinculada aos métodos e classes onde, seriam necessários executar os testes posteriores. Desta forma, a classe responsável pela execução dos testes, varreria as classes do sistemas e toda vez que encontra-se uma classe anotada por essa *interface*, executaria um teste, onde cada método que necessitaria de um teste possuiria em sua anotação o tipo do campo, as ações que deveriam ser executas e o valor que deveria ser preenchido.

Desta forma, os métodos da interface criada para esta solução, representa na verdade as ações que a classe `WebDrive` possibilita para os desenvolvedores.

3.5 Inserção das anotações no projeto

Por fim, a ultima etapa da solução em questão, trata-se de uma etapa contínua no projeto, pois sempre que uma nova classe é mapeada no sistema ou um novo cadastro é criado, a mesma ocorrerá. Nesta etapa se faz necessário a adição das anotações nos códigos do projeto, onde se deve inserir a anotação em todas as classes e métodos que devem ser testados. Na anotação são inseridas informações referente as ações que devem ser executadas para a propriedade em questão.

3.6 Avaliação da solução

Após a finalização do desenvolvimento desta primeira solução encontrada, notou-se a clara necessidade de alteração na sistemática de como os teste ocorreriam, necessidade esta oriunda de algumas limitações que a solução trás, como por exemplo, a não possibilidade de realizar testes que envolvam mais de um método, tornando praticamente inviável a tarefa de realizar testes funcionais, pois não seria possível mapear um cenário mais completo, que depende de mais de uma variável e que tem uma sequência bem mapeada que deve ser seguida.

Outro fator determinante para o descontinuamento desta solução, foi o fato de por utilizar anotações, o mesmo não possibilitaria o teste de um mesmo campo, com dois valores diferentes, pois só seria possível informar na anotação do método um valor de teste. Além disto, também não seria viável a execução de testes unitários para um sistema muito grande, pois é sabido que os testes devem ser executados sobre os pontos críticos do software e não na totalidade do projeto.

3.7 Códigos

3.7.1 `Teste.java`

A interface `Teste.java` procura com seus métodos mapear as ações que serão necessárias para a execução de um teste *Selenium* utilizando elementos da classe *WebDriver*, conforme apresentaremos mais adiante neste experimento.

Abaixo temos uma classe que demonstra a forma como ocorre as anotações utilizando a interface Teste.

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface Teste {
    String getUrl() default "";

    //findElement
    String getCampo() default ""; //campo html do formulário
    String getIdentificador() default "id"; //Informar se deve buscar um id, name, class ou
        css

    boolean isSelect() default false;

    String getValor() default ""; //utilizado como sendKeys e selectText
    boolean click() default false;
    boolean submit() default false;
    boolean limpar() default false;

    String getTipoAssert() default "igual";
    String getCampoAssert() default "";
    String getIdentificadorAssert() default "id";
    String getValorEsperadoAssert() default "";
    String getAtributoCampoComparacaoAssert() default "texto";

    boolean fazerLogin() default false;
    String getLogin() default "colegiado";
    String getSenha()default "";
}
```

Figura 3.2 – Interface Teste

3.7.2 `TesteFormulario.java`

Para realizar o teste propriamente dito, criou-se nos projetos utilizados, uma classe de testes denominada `TesteFormulaio.java`, onde a mesma utilizaria as anotações do *framework JUnit* para executar os testes necessários. A classe desenvolvida possui um método principal denominado `testaFormularios`, onde a ideia principal do mesmo seria buscar de forma recursiva, todas as classes do projeto anotadas pela interface Teste que criamos, e para cada uma das mesmas cria-se elementos *webDrive* para executar as ações indicadas pela anotação.

Para tornar o método `testaFormularios` e seus sub métodos mais genéricos, utilizou-se *Java Reflection* para tratar todas as classes e métodos simplesmente como objetos, possibilitando assim, o reuso do mesmo para qualquer teste necessário dentro deste projeto. Vemos, a seguir, o código desta classe:

```
@Teste(getUrl = "/cadastro-disciplina.htm", getCampo = "salvar", click = true
        ,getIdentificadorAssert = TestePropriedades.IDENTIFICADOR_CSS, getCampoAssert = "h4",
            getValorEsperadoAssert = "Sucesso!")
public class Disciplina {
    private Long id;
    private String codigo;
    private String nome;
    private Integer cargaHoraria;
    private Boolean preAprovada;
    private Boolean ativa;
    private Instituicao instituicao;

    @Teste(getCampo = "ativa1", click = true)
    public Boolean getAtiva() {
        return ativa == null || ativa;
    }

    public void setAtiva(Boolean ativa) {
        this.ativa = ativa;
    }

    @Teste(getCampo = "preAprovada1", click = true)
    public Boolean getPreAprovada() {
        return preAprovada != null && preAprovada;
    }

    public void setPreAprovada(Boolean preAprovada) {
        this.preAprovada = preAprovada;
    }

    @Teste(getCampo = "cargaHoraria", getValor = "60", isSelect = true)
    public Integer getCargaHoraria() {
        return cargaHoraria;
    }

    public void setCargaHoraria(Integer cargaHoraria) {
        this.cargaHoraria = cargaHoraria;
    }

    @Teste(getCampo = "nome", getValor = "Disciplina teste")
    public String getNome() {
        return nome;
    }
}
```

Figura 3.3 – Classe anotada com @Teste

```
@Test
public void testaFormularios() {
  for (Class classe : getCarregaClasses()) {
        Teste testeClasse = TestePropriedades.teste(classe);
        if (testeClasse.fazerLogin()) {
            LoginTeste.login(TestePropriedades.urlSistema, testeClasse.getSenha(),
                testeClasse.getLogin(), webDriver);
        }
        if (!testeClasse.getUrl().equals("")) {
            webDriver.get(TestePropriedades.urlSistema + testeClasse.getUrl());
        }
        for(Method metodo: classe.getDeclaredMethods()) {
            Teste teste = TestePropriedades.teste(metodo);
            if (teste != null) {
                executaTeste(teste, true);
            }
        }
        executaTeste(testeClasse, false);
        System.out.println("Formulário da classe " + classe.getName() + " testado!");
  }
}
```

4 PRÓXIMAS ETAPAS

Como continuação do trabalho realizado até o momento, são planejadas as seguintes atividades:

REFERÊNCIAS

BIASI, L. B. Geração automatizada de drivers e stubs de teste para JUnit a partir de especificações U2TP., [S.l.], 2006.

BOEHM, B. W.; BROWN, J. R.; LIPOW, M. Quantitative evaluation of software quality. In: SOFTWARE ENGINEERING, 2. **Proceedings...** [S.l.: s.n.], 1976. p.592–605.

CAGNIN, M. I. et al. Reuso na Atividade de Teste para Reduzir Custo e Esforço de VV&T no Desenvolvimento e na Reengenharia de Software. **XVIII Simpósio Brasileiro de Engenharia de Software (SBES)**, [S.l.], p.71–84, 2004.

CHIAVEGATTO[1], R. B. et al. Desenvolvimento Orientado a Comportamento com Testes Automatizados utilizando JBehave e Selenium., [S.l.], 2013.

CRESPO, A. N. et al. Uma metodologia para teste de Software no Contexto da Melhoria de Processo. **Simpósio Brasileiro de Qualidade de Software**, [S.l.], p.271–285, 2004.

GARVIN, D. A. Competing on the 8 dimensions of quality. **Harvard business review**, [S.l.], v.65, n.6, p.101–109, 1987.

GUIZZARDI, G. **Desenvolvimento para e com reuso**: um estudo de caso no domínio de vídeo sob demanda. 2000. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Espírito Santo.

KOSCIANSKI, A.; SOARES, M. d. S. **Qualidade de software**. [S.l.]: São Paulo: Novatec Editora, 2007.

LOPES, D. **O Cucumber ainda tem o seu valor**. Accessed: 2015-08-10, http://www.infoq.com/br/articles/valor-cucumber.

MALDONADO, J. C. et al. Introduçao ao teste de software. **São Carlos**, [S.l.], 2004.

NUNES, D. Automação de Testes de Aceitação com Cucumber e JRuby., [S.l.], 2009.

PEREIRA, D. V. Estudo da ferramenta selenium ide para testes automatizados de aplicações web., [S.l.], 2012.

PRESSMAN, R. S. **Engenharia de software**. [S.l.]: McGraw Hill Brasil, 2011.

REFERENCE Cucumber.io. Accessed: 2015-08-05, https://cucumber.io/docs/reference.

RIOS, E. **Teste de software**. [S.l.]: Alta Books Editora, 2006.

SCHMITZ, F. H.; BECKER, H.; BERLATTO, L. d. F. **Cucumber - Um breve review**. Accessed: 2015-08-06, http://pt.slideshare.net/LaisBerlatto2/cucumber-um-breve-review.

SIXPENCE, E.; ADÃO, P.; SMITH, C. AUTOMATIZAÇÃO DE CASOS DE TESTE COMO PROCESSO DE MELHORIA DA QUALIDADE DO SOFTWARE: o caso da aplicação e-learning isupac3 no isutc. In: CONGRESSO LUSO MOçAMBICANO DE ENGENHARIA - CLME. **Anais...** [S.l.: s.n.], 2011.

VASCONCELOS, A. M. L. de et al. Introdução à Engenharia de Software e à Qualidade de Software., [S.l.], 2006.