CENTRO UNIVERSITÁRIO DO SUL DE MINAS - UNIS-MG

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

MARINA ADRIANA FARIA DE MORAES

AUTOMAÇÃO DE TESTES DE REGRESSÃO E *SMOKE TEST* EM APLICAÇÕES *WEB* COM O *SELENIUM*

Varginha/MG
2013

MARINA ADRIANA FARIA DE MORAES

AUTOMAÇÃO DE TESTES DE REGRESSÃO E *SMOKE TEST* EM APLICAÇÕES *WEB* COM O *SELENIUM*

Trabalho apresentado ao curso de Bacharelado em Sistemas de Informação do Centro Universitário do Sul de Minas – UNIS/MG como pré-requisito para obtenção do grau de bacharel, sob orientação do(s) Prof.(s) Weider Pereira Rodrigues.

Varginha/MG
2013

MARINA ADRIANA FARIA DE MORAES

AUTOMAÇÃO DE TESTES DE REGRESSÃO E *SMOKE TEST* EM APLICAÇÕES *WEB* COM O *SELENIUM*

Monografia apresentada ao curso de Bacharelado em Sistemas de Informação do Centro Universitário do Sul de Minas – UNIS/MG, como pré-requisito para obtenção do grau de bacharel pela Banca Examinadora composta pelos membros:

Aprovado em / /

_____

Prof. Esp. Weider Pereira Rodrigues

_____

Prof. Esp. César Fernandes Ribeiro Filho

_____

Prof. Esp. Rodrigo Gomes da Silva

OBS.:

AGRADECIMENTOS

Agradeço a Deus, ao meu noivo Fernando e à minha família por acreditarem e me apoiarem nestes quatro anos de faculdade. Aos meus professores, em especial ao prof. Weider, pelo incentivo e orientação na construção deste trabalho.

“A persistência é o menor caminho do êxito”.
(Charles Chaplin)

RESUMO

As empresas que trabalham com desenvolvimento de software estão cada vez mais interessadas em melhorar a qualidade de seus produtos e serviços e o setor de qualidade das empresas desempenha papel importante, realizando atividades que aumentam a qualidade do produto, através de diversos testes que são realizados. Entretanto, alguns tipos de testes são considerados maçantes, podendo ser automatizados com o auxílio de ferramentas disponíveis no mercado. Recentemente, a empresa *Web*Aula implantou nos processos do setor de qualidade, o projeto de automação de testes. Testes como os de regressão e *smoke test* foram automatizados, proporcionando aos testadores mais tempo para se dedicar a atividades que necessitam ser executadas manualmente. Este trabalho tem como objetivo avaliar as vantagens e desvantagens em se automatizar testes de regressão e *smoke test* na empresa, bem como apontar os benefícios que a *Web*Aula já alcançou desde a implantação do projeto, podendo assim julgar se é viável ou não a automação de testes considerados repetitivos na organização.

Palavras-chave: Automação de testes. Regressão. *Smoke test*.

ABSTRACT

Companies who work with software development are increasingly interested in improving the quality of its products and services and industry enterprise quality plays an important role , performing activities that increase the quality of the product through various tests that are performed . However, some types of testing are considered dull and may be automated with the aid of tools available on the market. Recently, the company WebAula deployed in industry processes , quality design automation testing . As regression testing and smoke test were automated , giving testers more time to devote to activities that need to be performed manually . This study aims to evaluate the advantages and disadvantages to automate regression testing and smoke test in the company , as well as point out the benefits that WebAula already achieved since the implementation of the project , so can judge whether or not it is feasible to test automation considered repetitive organization .

Keywords: *Test automation. Regression. Smoke test.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

A informatização está expandindo cada vez mais pelo mundo, desde pequenos até grandes empreendimentos possuem um computador e pelo menos um sistema básico para atender suas necessidades.

A *Web*Aula é uma empresa que atua no mercado a mais de 10 anos oferecendo produtos e serviços no segmento *e-Learning* (Educação a Distância). O LMS (*Learning Management System*), seu principal produto, já treinou mais de cinco milhões de pessoas entre alunos e funcionários, sendo utilizado por instituições de ensino, empresas privadas e órgãos governamentais, permitindo que o conhecimento seja difundido entre diversas pessoas a qualquer hora e em qualquer lugar.

O LMS é um sistema que permite, dentre seus diversos recursos, que gestores disponibilizem aos alunos cursos no padrão *SCORM*[1], avaliações, trabalhos, exercícios, certificados de conclusão e aproveitamento. Permite a interação entre os alunos, tutores e gestores através dos recursos comunidade, *chat*, fórum e central de mensagens para envio e recebimento de *e-mails*.

Mercado aquecido é sinônimo de oportunidades, com isso, surgem diariamente novas empresas que desenvolvem produtos de *software*. Com o aumento da concorrência, a necessidade de investir em qualidade para ganhar mercado frente aos concorrentes é grande.

Uma forma de se garantir a qualidade de um *software* é testando-o, e a empresa *Web*Aula, com o intuito de melhorar a qualidade de seus produtos e garantir a satisfação do cliente, iniciou um projeto de automação de testes e investiu na contratação de profissionais qualificados para o projeto, paralelo as atividades de testes manuais já executadas nos produtos da empresa.

Existem diversos tipos de testes, seja para garantir seu funcionamento conforme especificado pelo cliente, seja a validação de interface, segurança ou performance do sistema. O teste é uma atividade que nunca cessa, sempre que são realizadas novas atualizações do sistema, sejam correções ou melhorias, tem-se a necessidade de realização de testes nas novas atualizações e também nas funcionalidades já existentes a fim de garantir que não foram inseridos novos *bugs*[2] no sistema.

---

[1] Modelo de referência padronizado com especificações utilizadas mundialmente em cursos *web*.
[2] São erros encontrados em um *software* ou *hardware*, que interferem no seu correto funcionamento.

O trabalho de teste que é considerado maçante e repetitivo como, por exemplo, testes de regressão e *smoke test*, podem ser automatizados com o auxilio de uma ferramenta que cria *scripts*[3]. A ferramenta permite que os *scripts* sejam gravados e executados sempre que necessário, realizando o trabalho repetitivo e que não necessita de intervenção manual do testador.

A tendência é que, futuramente, a automação de testes esteja inserida nos processos de grande parte das empresas de desenvolvimento de *software*, pois auxilia na busca constante pela qualidade do produto e, consequentemente, garante a satisfação dos usuários.

1.1 Assunto e tema específico

Automação de testes de regressão e *smoke test* em aplicações *web* com o *Selenium*

1.2 Problema de pesquisa

Automatizar testes de regressão e *smoke test* é mais viável que realizá-los manualmente?

1.3 Hipóteses

- A automação de testes pode aumentar a cobertura dos testes no sistema, agregando mais qualidade ao produto.
- Os profissionais da área de testes terão mais tempo disponível para realizar atividades que precisam ser executadas manualmente.

[3] Conjunto de instruções em determinada linguagem de programação que quando executadas geram ações em um *software*.

- A tendência é que, à longo prazo, a automação de testes reduza custos na empresa, já que grande parte dos erros poderão ser encontrados e corrigidos antes mesmo que os clientes os veja em produção.

1.4 Objetivo geral

Mostrar que a automação de testes de regressão e *smoke test* é um investimento benéfico que já produz resultados positivos para a empresa *Web*Aula.

1.5 Objetivos específicos

- Mostrar *scripts* de automação de testes com o *Selenium*.
- Comparar o tempo gasto por um profissional para executar os testes manuais e o tempo de execução dos mesmos testes, porém, automatizados.
- Levantar a quantidade de chamados abertos antes e depois da automação de testes.
- Calcular o tempo que cada profissional terá livre a curto, médio e longo prazo, analisando o pior e o melhor cenário quanto à execução de testes de regressão e *smoke test*.
- Analisar as vantagens obtidas pela empresa *Web*Aula ao automatizar testes de regressão e *smoke test*.

1.6 Justificativa

Devido à necessidade crescente das organizações em se equipar com as novas tecnologias, cresce cada vez mais a busca por diversos tipos de *softwares* no mercado. Para atender esta demanda, além das já existentes, surgem novas organizações que trabalham com desenvolvimento de *softwares*. Com o aumento da concorrência em um mercado tão

diversificado, é preciso além de oferecer produtos que atendam as necessidades dos usuários, investir em um *software* de qualidade para ganhar cada vez mais mercado.

Para garantir maior qualidade ao *software*, é preciso iniciar as verificações e validações que se fizerem necessárias desde o início do ciclo de vida do *software* e não somente no final do ciclo, pois irá gerar maior retrabalho e mais custos para o projeto quando os defeitos são identificados no final do processo.

Atualmente há no mercado profissionais que realizam a validação nas aplicações desenvolvidas através de diversos tipos de testes, sendo o teste funcional o mais importante e mais utilizado, pois é através dele que é possível validar se o que foi desenvolvido está de acordo com as necessidades do usuário.

Mesmo após validar o sistema e entregá-los aos clientes, o trabalho de validação nunca cessa, pois surgem novas necessidades das organizações ou até mesmo correções, que levam a novos problemas, sendo necessária uma validação constante a fim de manter a qualidade do produto.

Hoje, no mercado, existem diversas ferramentas, *open source* ou proprietárias, que auxiliam os profissionais de qualidade nesta validação. *Scripts* de automação são criados para executar o teste pesado, rotineiro, que sempre é feito pelo profissional, garantido que os recursos do sistema estão funcionando.

A automação de testes proporciona benefícios como a rápida execução de um grande volume de casos de testes, possibilidade de se repetir estes casos de testes exaustivamente a qualquer hora do dia ou da noite, além de aumentar a disponibilidade de outros profissionais da equipe para tarefas que exigem a execução manual.

De acordo com Molinari (2008a) a automação:

> é um investimento que será pago na economia de problemas.[...] Num primeiro momento, a automação/gravação dos scripts com ações de testes levam mais tempo, mas este mesmo tempo gasto será plenamente recuperado (e com muito lucro) ao executar os testes automatizados.

Automatizar testes é um projeto que demanda tempo durante o planejamento e criação dos *scripts* e fica mais complexa quando se quer aumentar a abrangência dos testes, quanto mais elaborado for o *script* de automação, maior será sua cobertura de teste.

2 REVISÃO DA LITERATURA

Neste capítulo será fundamentada a visão de diversos autores sobre o processo de desenvolvimento de *software*, metodologias utilizadas, a importância dos testes de *software* como garantia de qualidade bem como a importância da automação de testes de regressão e *smoke test* nas organizações.

2.1 Software x hardware

Softwares são programas inseridos dentro de um *hardware* com a finalidade de realizarem determinadas tarefas. O *hardware* é a parte física do computador e depende do *software*, que é a parte lógica, para funcionar.

Os *softwares* são classificados em *Software* Básico, Aplicativos e Utilitários. Os *softwares* básicos são sistemas operacionais, compiladores, linguagens de programação, que permitem que o usuário e a máquina se comuniquem através de uma interface gráfica. Os *softwares* aplicativos são aqueles que auxiliam os usuários diariamente, como navegadores, editores de textos. Os *softwares* utilitários são os que possibilitam ao usuário realizar tarefas não inseridas ao sistema operacional, como zipar arquivos, desfragmentar unidades de disco.

2.1.1 *Software* proprietário, s*oftware* livre e *open source*

Softwares proprietários são aqueles que necessitam de licenças, muitas vezes onerosas ou da autorização do proprietário para que sejam feitas cópias, redistribuição ou modificações no *software*.

Para Stallman (1983 *apud* CAMPOS, 2006, p.1) software livre é:

> o *software* que pode ser usado, copiado, estudado, modificado e redistribuído sem restrição. A forma usual de um *software* ser distribuído livremente é sendo acompanhado por uma licença de *software* livre (como a GPL ou a BSD), e com a disponibilização do seu código-fonte. Um *software* é tido como livre quando atende

> aos quatro tipos de liberdade para os usuários: Liberdade 0: A liberdade para executar o programa, para qualquer propósito; Liberdade 1: A liberdade de estudar o *software*; Liberdade 2: A liberdade de redistribuir cópias do programa de modo que você possa ajudar ao seu próximo; Liberdade 3: A liberdade de modificar o programa e distribuir estas modificações, de modo que toda a comunidade se beneficie.

O s*oftware* livre normalmente é gratuito, mas não necessariamente, ou seja, pode ser vendido desde que as liberdades apontadas pela Free Software Foundation sejam respeitadas.

Open source (código aberto):

> "não significa apenas acesso ao código-fonte. Os termos de distribuição de software open-source devem cumprir os seguintes critérios: Redistribuição livre; Código - Fonte; Obras Derivadas; Integridade do Código Fonte do Autor; Sem Discriminação Contra Pessoas ou Grupo; Sem Discriminação Contra Campos de Trabalho; Distribuição da Licença; Licença não deve ser específica para um Produto; Licença não deve restringir Outro Software; Licença deve ser tecnologicamente neutra." (DEFINIÇÃO...p.1)

Os *softwares* proprietário, livre ou código aberto possuem suas vantagens e desvantagens. O administrador deve traçar o perfil da empresa, bem como suas reais necessidades e analisar qual *software* irá suprir estas necessidades visando melhorar os processos dentro da organização.

2.2 O processo de s*oftware*

O processo de *software* é composto por diversas atividades organizadas com o propósito de produzir um produto de *software*. É considerado um dos principais meios que tem como propósito obter um *software* de qualidade dentro da Engenharia de *Software*.

De acordo com Schwartz (1975, p.7), as principais fases de um processo de *software* são:

> Especificação de requisitos: tradução da necessidade ou requisito operacional para uma descrição da funcionalidade a ser executada. Projeto de sistema: tradução destes requisitos em uma descrição de todos os componentes necessários para codificar o sistema. Programação (Codificação): produção do código que controla o sistema e realiza a computação. Verificação e Integração (*Checkout*): verificação da satisfação dos requisitos iniciais pelo produto produzido.

Para cada fase do processo de *software*, há um grupo de atividades a serem cumpridas com intuito de atingir o objetivo proposto. Na especificação constam as atividades de engenharia de sistema, análise de requisitos e especificação de sistema. No projeto, as atividades são projeto arquitetural, projeto de interface e projeto detalhado. Na implementação está a codificação, ou seja, a implementação em si do sistema em uma linguagem de computador. Na fase de validação estão testes de unidade e módulo e integração. E por último, a fase manutenção e evolução, onde o *software* entra em um ciclo frequente que envolve todas as fases anteriores. (PRESSMAN, 1997; SCHWARTZ, 1975; SOMMERVILLE, 1995).

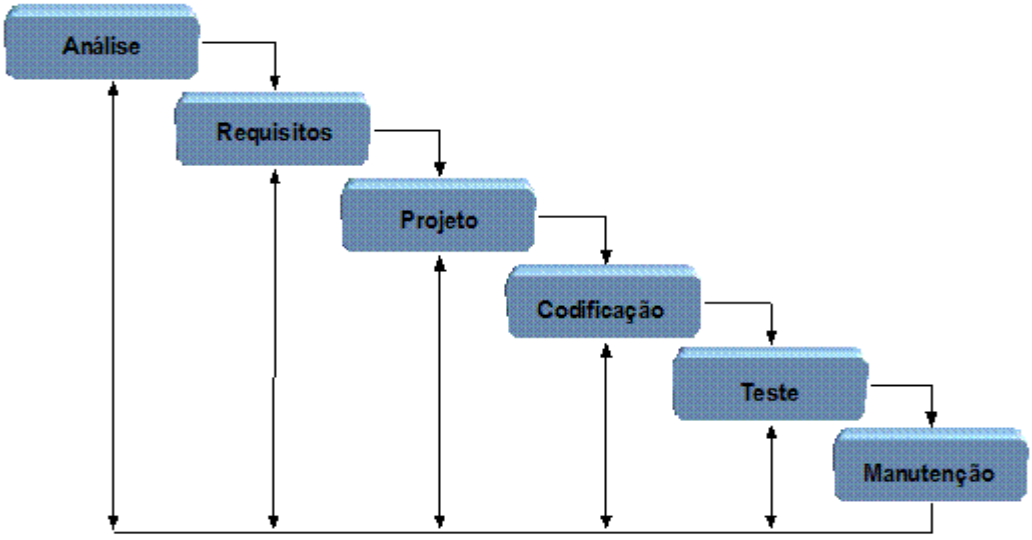
As atividades descritas acima são os passos básicos que devem ser seguidos para se obter um produto de *software*. A ordem de execução destas atividades é definida de acordo com a metodologia de desenvolvimento de *software* utilizada pelas organizações.

2.3 Metodologias de desenvolvimento de s*oftware*

Metodologia de desenvolvimento é um conjunto de práticas recomendadas para o desenvolvimento de *softwares*, sendo que essas práticas, geralmente, passam por fases ou passos, que são subdivisões do processo para ordená-lo e melhor gerenciá-lo (SOMMERVILLE, 2000).

O modelo cascata, também conhecido como clássico ou linear, é o mais utilizado entre os modelos de desenvolvimento tradicionais. A Figura 01 exemplifica as fases deste modelo, que foi descrito pela primeira vez por Winston W. Royce no ano de 1970.

Figura 01 – Modelo Cascata



Fonte: PRESSMAN (2006).

Este modelo tem como principal característica a sequencialidade de atividades, ou seja, o início da próxima fase só se dá quando há o término da fase anterior. Os testes só ocorrem na fase final do processo, o que influencia negativamente na qualidade do produto e gera custos para ajustar erros de um sistema inteiro, podendo esta correção gerar novos problemas em outros pontos do sistema.

O *software* é desenvolvido em um longo prazo e a entrega do produto ao usuário só ocorre quando finalizar todo o processo. Como o usuário não acompanha o processo de desenvolvimento, caso o sistema esteja diferente do solicitado e o usuário não o aprove, o *software* deverá ser refeito, gerando mais custo, e mais tempo tanto dos profissionais envolvidos, quanto da espera do usuário para o recebimento do produto.

O *Scrum* é uma metodologia ágil usada na gestão e planejamento de projetos de *software*. Nesta metodologia um projeto é dividido em ciclos, onde são realizados simultaneamente todos os processos de desenvolvimento de *software* em cada ciclo. Como a validação (testes) ocorre em paralelo ao desenvolvimento, o retrabalho durante as correções é bem menor, o que gera menor custo com o projeto. Nesta metodologia, conforme exemplificado na figura 02, podem ser realizadas entregas parciais do sistema ao usuário, que começa a receber algumas funcionalidades do *software* em intervalos menores de tempo e ainda consegue validar se a parte entregue está como solicitado. Caso seja necessária alguma alteração, a correção se torna mais rápida, gera menos custo e não compromete todo o sistema.

Figura 02 - O ciclo de vida do Scrum

Fonte: THAMIEL (2009).

2.4 Qualidade de s*oftware*

Já faz algum tempo que a busca pela qualidade não é mais um diferencial e sim um pré-requisito para a empresa atuar com sucesso no mercado, em empresas que trabalham com desenvolvimento de *software* não é diferente. A engenharia de *software* auxilia as empresas, pois ambas tem o mesmo objetivo que é oferecer *software* de qualidade. Segundo Filho (2008, p.1):

> A Engenharia de *Software* visa à criação de produtos de *software* que atendam as necessidades de pessoas e instituições e, portanto, tenham valor econômico. Para isso, usa conhecimentos científicos, técnicos e gerenciais, tanto teóricos quanto empíricos. Ela atinge seus objetivos de produzir *software* com alta qualidade e produtividade quanto é praticada por profissionais treinados e bem informados, utilizando tecnologias adequadas, dentro de processos que tirem proveito tanto da criatividade quando da racionalização do trabalho.

Para uma visão mais ampla sobre qualidade, segue abaixo alguns conceitos definidos por especialistas no assunto:

> A qualidade pode ser medida através do grau de satisfação em que as pessoas avaliam determinado produto ou serviço. No entanto, esse produto ou serviço pode

> ter qualidade para algumas pessoas e para outras nem tanto, ou seja, a qualidade é algo subjetivo. Conceituar o termo qualidade se torna uma tarefa muito difícil, pois elementos intrínsecos estão enraizados no intelecto de cada ser. (CAMPOS, 2008, p.1)

Segundo Bartié (2002, p.16), qualidade de *software* é "[...] um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos de produtos, prevendo e eliminando defeitos".

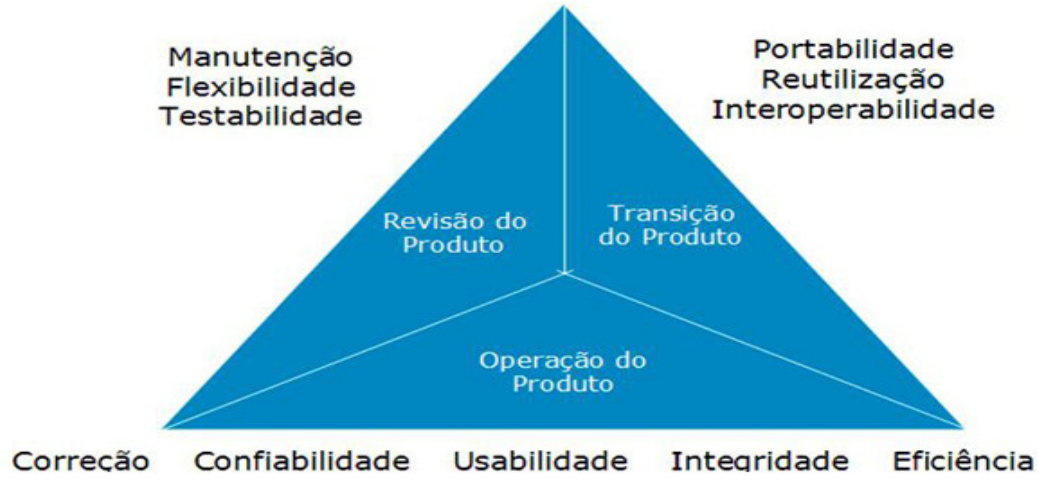
De acordo com Rocha(1987, p.1), qualidade:

> é um conceito multidimensional, realizando-se por intermédio de um conjunto de atributos ou características. As empresas responsáveis pelo desenvolvimento de *software* devem assumir a responsabilidade de estabelecer este nível aceitável de qualidade e meios para verificar se ele foi alcançado. Qualidade de *software* é, portanto, um conjunto de propriedades a serem satisfeitas, em determinado grau, de modo que o *software* satisfaça as necessidades de seus usuários.

De acordo com Caetano (2007, p.1) "desenvolver software de qualidade não é mais um requinte para poucos, transformou-se num fator de competitividade num mercado cada vez mais exigente".

Conforme pode ser observado na figura 03, sugere-se como métricas de qualidade a avaliação de três pontos distintos: Revisão do Produto, Transição do Produto e Operação do Produto (MCCALL e CAVANO, 1978 *apud* CAMPOS, 2008, p.1).

Figura 03 - Fatores da Qualidade



Fonte: CAMPOS (2008)

A qualidade faz parte do dia a dia e desempenha papel importante em toda e qualquer organização. Durante muito tempo, os testes de softwares eram efetuados pelos próprios desenvolvedores (testes unitários), no entanto, as informações extraídas destes testes não permitiam que todos os defeitos fossem detectados. Além dos desenvolvedores, os próprios usuários do sistema eram envolvidos para aprovar o resultado desses testes ou mesmo participar da criação dos dados de teste, gerando uma incidência muito grande de defeitos, que só apareciam quando os sistemas já estavam em produção, consequentemente, sua correção se tornava muito mais cara. (BASTOS, 2007).

Com isso, as organizações perceberam a necessidade de mudança nos processos com intuito de melhorar a qualidade do *software* desenvolvido. As empresas estão aprimorando as atividades de teste, investindo na contratação de profissionais especializados, que em parceria com os demais integrantes do projeto, atuam durante todo o ciclo de vida do *software* a fim de garantir mais qualidade ao produto entregue.

2.5 Testes de s*oftware*

Em um projeto, 40% do tempo planejado é destinado a fase de teste, um defeito descoberto tardiamente provoca um acréscimo de 60% nos custos do projeto (PRESSMAN, 2006). Este é um dos motivos de o teste ser considerado uma das atividades mais importantes durante o desenvolvimento de um *software*.

De acordo com ISTQB (2011, p.11):

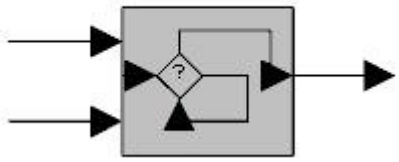
> o ser humano está sujeito a cometer um erro (engano), que produz um defeito (falha, bug) no código, em um software ou sistema ou em um documento. Se um defeito no código for executado, o sistema falhará ao tentar fazer o que deveria (ou, em algumas vezes, o que não deveria), causando uma falha. Defeitos no software, sistemas ou documentos resultam em falhas, mas nem todos os defeitos causam falhas.

Segundo Hetzel (1987, p.6), “teste é qualquer atividade que vise avaliar uma característica ou recurso de um programa ou sistema.” O teste além de detectar problemas no *software*, garante que as funcionalidades do sistema estão de acordo com as solicitações do usuário.

De acordo com Molinari (2008b), casos de teste "são as situações de teste em si, em nível detalhado, podendo abranger um ou mais requisitos de teste[4]." Testar um *software* não significa apenas gerar e executar casos de teste. É preciso planejamento, gerenciamento e análise de resultados (NETO, 2008). Ao realizar testes é possível medir a qualidade do produto que será entregue ao usuário.

A figura 04 mostra a técnica de teste estrutural ou de caixa branca, que tem como objetivo identificar defeitos na estrutura interna do produto. Essa técnica trabalha diretamente sobre o código fonte do componente de *software* para avaliar aspectos tais como: teste de condição, teste de fluxo de dados, teste de ciclos e teste de caminhos lógicos (PRESSMAN, 2005).

Figura 04 - Técnica de Teste Estrutural



Fonte: NETO (2008).

Outra técnica é o teste de caixa-preta, mostrado na figura 05, também conhecido como teste funcional é baseado nos requisitos funcionais do *software*, a fim de garantir que os requisitos foram total ou parcialmente obtidos.

Figura 05 - Técnica de Teste Funcional

Fonte: NETO (2008)

De acordo com Inthurn (2001, p.56), o teste funcional procura descobrir "funções incorretas ou ausentes; erros de interface; erros nas estruturas de dados ou no acesso a bancos de dados externos; erros de desempenho; erros de inicialização e término".

[4] O que se quer testar, ou seja, o(s) objetivo(s) do teste.

Erros comuns de dados de interface que são encontrados e podem ser citados são sistemas que permitem inserção de datas de nascimento futuras, campos de preenchimento obrigatórios vazios ou com espaços brancos ou zeros, letras e caracteres especiais em campos que só permitem números. É importante que o testador simule e identifique este tipo de erro, pois eles podem ser cometidos por usuários até mesmo com o objetivo de burlar o sistema.

2.5.1 *Smoke test* ou teste de fumaça

De acordo com McConnell (1996, p.73), *Smoke Teste* "é o processo no qual um produto de *software* é diariamente montado e submetido a uma série de testes para verificar sua funcionalidade básica." Quando um novo sistema é liberado para testes ou quando são inseridas alterações em seus componentes, são realizados testes básicos a fim de identificar erros "bloqueadores" nas funcionalidades que causariam maior prejuízo ao usuário.

Sendo identificadas falhas nestas funcionalidades, o pacote a ser testado retorna para a equipe de desenvolvimento para as devidas correções antes mesmo de acabar o processo de teste em todo o sistema. Ao retornar para testes, novamente deve-se realizar o teste de fumaça, para se garantir o funcionamento básico do produto, caso não sejam encontrados novos *bugs*, outras funcionalidades do sistema serão testadas.

2.5.2 Testes de regressão

Os testes de regressão são executados sempre que ocorrem correções ou são inseridas novas funcionalidades ao sistema, a fim de garantir que novos *bugs* não foram acrescentados às funcionalidades já existentes no produto. A execução de testes de regressão absorve muito tempo e se torna cansativa, além de ter custo elevado, uma vez que devem ser executados a cada nova atualização do *software*.

O teste de *software* é uma fase trabalhosa e que demanda muito dinheiro durante o processo de *software*, por isso estão entre as primeiras ferramentas de *software* a serem

desenvolvidas. Elas oferecem uma variedade de recursos e seu uso pode reduzir significativamente os custos de teste (SOMMERVILLE, 2007).

Executar apenas testes de regressão manualmente nem sempre é viável, pois devido à extensão dos testes, muitas vezes não é possível realizar a validação em todo o sistema, sendo necessário priorizar os itens que são considerados mais críticos. Consequentemente, os itens que não foram testados, com certeza possuem algum tipo de erro que não será solucionado enquanto o item não for considerado crítico.

2.6 Automação de testes

Automatizar testes significa realizá-los com o apoio de ferramentas apropriadas. São testes que necessitam de alguém para implementar, ou seja, criar os *scripts* que serão executados de forma rápida, a qualquer hora do dia, quantas vezes for necessário, minimizando os esforços manuais repetitivos durante a realização dos testes.

Um dos motivos que deve ser levado em consideração ao decidir por automatizar ou não os testes de um sistema, é a necessidade constante de re-execução dos mesmos. Testes que são executados diversas vezes manualmente, poderão ser executados em menos tempo, reduzindo o esforço de teste e permitindo a re-execução sempre que necessário.

Dois outros pontos importantes a serem observados são a automação de funcionalidades consideradas críticas, que podem ser testadas exaustivamente com os *scripts* a fim de evitar grande quantidade de *bugs*, garantindo a estabilidade da aplicação. *Softwares* que estão prestes a mudar de interface, por exemplo, não devem ser automatizados, pois o tempo e esforço gastos na criação de *scripts* serão perdidos.

Existem no mercado diversas ferramentas de automação de teste, mas, para toda e qualquer organização, a melhor ferramenta é aquela que atende grande parte das suas necessidades.

Entretanto, de nada adianta ter uma boa ferramenta nas mãos e não souber manipulá-la. Para que a automação de testes seja bem sucedida e traga lucros e benefícios para as empresas, é preciso que os profissionais envolvidos no projeto tenham conhecimento em

testes de *software*, conheçam o sistema e suas regras de negócio, além é claro de conhecer a ferramenta de automação e a linguagem de programação que será utilizada.

Muitos projetos de automação de testes fracassam porque os envolvidos têm visões diferentes sobre os benefícios e limitações do projeto. A gerência acredita que automatizar irá sanar todos os problemas. Os desenvolvedores desconhecem a necessidade de criar meios para aumentar a abrangência dos testes durante o desenvolvimento. Os testadores acreditam que automatizar é transformar testes manuais em *scripts*, sendo considerada por eles uma tarefa simples de ser realizada. (CAETANO, 2008).

A empresa precisa planejar e detalhar todo o projeto de automação, para que não ocorram imprevistos que podem causar danos irreversíveis, ocasionando o fracasso do projeto.

2.7 *Selenium*

O *Selenium* é um conjunto de ferramentas *open source* (código aberto), que permite criar *scripts* de testes funcionais automatizados para aplicações *Web* e podem ser executados em qualquer navegador que possui suporte a *JavaScript*. Dentre suas principais ferramentas, serão descritas abaixo o *Selenium IDE* e *Selenium RC*.

2.7.1 *Selenium IDE*

O *Selenium IDE* é um *plug-in* do navegador *Mozilla Firefox* do tipo *record-and-playback*. Depois de instalado, o *Selenium IDE* poderá ser acessado no menu Ferramentas do navegador. Ao clicar no botão Gravar, todas as ações realizadas pelo testador durante a navegação no sistema *web* serão capturadas. Será gerado um *script* que poderá ser salvo e reexecutado sempre que se fizer necessário. Uma grande vantagem no uso desta ferramenta é o fato de que o testador não precisa ter conhecimento em programação para automatizar testes.

2.7.2 *Selenium RC*

No *Selenium RC* é possível converter os *scripts* criados pelo *Selenium IDE* para uma das linguagens de programação suportadas por ele (HTML, Java, C#, Perl, PHP, *Python*, e *Ruby*), permitindo que sejam inseridas lógicas de programação complexas possibilitando uma validação mais abrangente das regras de negócios existentes no sistema.

3 MATERIAL E MÉTODOS

De acordo com Mascarenhas (2012, p.35) a metodologia serve para "explicar tudo o que foi feito durante um estudo. O objetivo é descrever o método, os participantes, o tipo de pesquisa e os instrumentos utilizados [...]". " Em geral, chamamos de método o conjunto de técnicas que usamos em um estudo para obter uma resposta. Em outras palavras, ele é o caminho que percorremos para chegar a uma conclusão científica" (id., 2012, p.36).

Como o objetivo da pesquisa era apontar os benefícios que a empresa *Web*Aula está obtendo com a implantação do projeto de automação de testes no setor de qualidade da empresa, inicialmente foi realizada uma pesquisa bibliográfica sobre qualidade de *software*, testes de regressão, *smoke test* e automação de testes. A *Web*Aula utiliza a ferramenta *Selenium* para a automação de testes, com isso, foi realizada uma pesquisa sobre o *Selenium* e suas funcionalidades.

Em um segundo momento, foi feita uma entrevista com o coordenador da área de qualidade da empresa *Web*Aula, conforme é possível verificar no "Anexo A". O intuito era saber quais as necessidades existiam no setor e que motivaram o início do projeto de automação de testes, como está o projeto e quais resultados já foram alcançados desde sua implantação.

Posteriormente, foi realizada uma análise do dia a dia dos testadores dentro da empresa. A empresa não segue rigorosamente nenhuma metodologia de desenvolvimento, entretanto, conseguiu adaptar ao seu cotidiano alguns fundamentos do *Scrum.*

Para mensurar o tempo que o testador levava para realizar o *smoke test* e mini regressão no LMS, foi feito um acompanhamento de sua rotina se baseando no pior e melhor cenário já vivenciados pela empresa, *build*[5] aplicado todos os dias e a cada 15 dias, respectivamente, a fim de descobrir quanto tempo o testador destinava a cada *build* para esta atividade. O mesmo foi feito quanto ao teste de regressão após a aplicação do *build* em produção. Foi cronometrado ainda quanto tempo se leva para executar os *scripts* automatizados dos mesmos testes realizados manualmente pelo testador.

Foi realizado também um levantamento sobre a quantidade de chamados que a equipe de qualidade mantinha aberto antes e quantos novos foram criados após a automação de testes ter sido implantada no setor.

[5] Pacote de correções e/ou melhorias gerado para que o setor de qualidade da empresa teste e valide se estão aptos para serem aplicados nos ambientes de produção.

De posse destes dados, foi possível avaliar os benefícios que a *Web*Aula já obteve e ainda obterá com a inserção da automação de testes nos processos da empresa.

O foco deste projeto não era realizar apontamentos ou comparações sobre custo e tempo gastos para criação dos *scripts* de automação de testes, uma vez que o tempo de criação e cobertura desejada bem como a complexidade do *script*, iriam variar a cada caso de teste, o que interferiria diretamente na tentativa de mensurar tais dados.

4 RESULTADOS

Neste capítulo serão apresentados os dados colhidos no setor de qualidade sobre os testes de regressão e *smoke test* por eles realizados a cada aplicação de *build*. Em seguida, será realizada a análise destes dados, a fim de se confirmar ou negar as hipóteses descritas neste trabalho, bem como avaliar se o objetivo deste foi atingido.

4.1 Levantamento de dados

O setor de qualidade da empresa, o QA, possui fluxos para todo *build* aplicado em homologação. Sempre que eram liberados itens a serem homologados, o banco de dados era recriado e os testadores precisavam inserir novos dados no sistema. As principais telas do LMS da *Web*Aula foram inseridas no "Anexo B".

Para realizarem os testes no sistema, os testadores precisavam realizar alguns cadastros, muitas vezes considerados massa de dados para se chegar a determinado teste. Entretanto, em alguns testes, nem sempre havia a necessidade de realizar todos os cadastros básicos. Por exemplo, quando o teste era cadastrar aluno no sistema, era preciso inserir apenas um segmento (massa de dados) e um ou mais alunos para testar o cadastro. Desta forma, em alguns *builds*, o *smoke test* era realizado parcialmente no sistema.

Após a homologação de cada *build* e aplicação em produção, que acontecia a cada 15 dias, era preciso ainda realizar testes de regressão em produção para garantir que nenhum novo erro foi inserido nas diversas funcionalidades do sistema.

Os dados levantados quanto ao *smoke test* e regressão realizados manualmente pelo testador serão descritas na tabela 01:

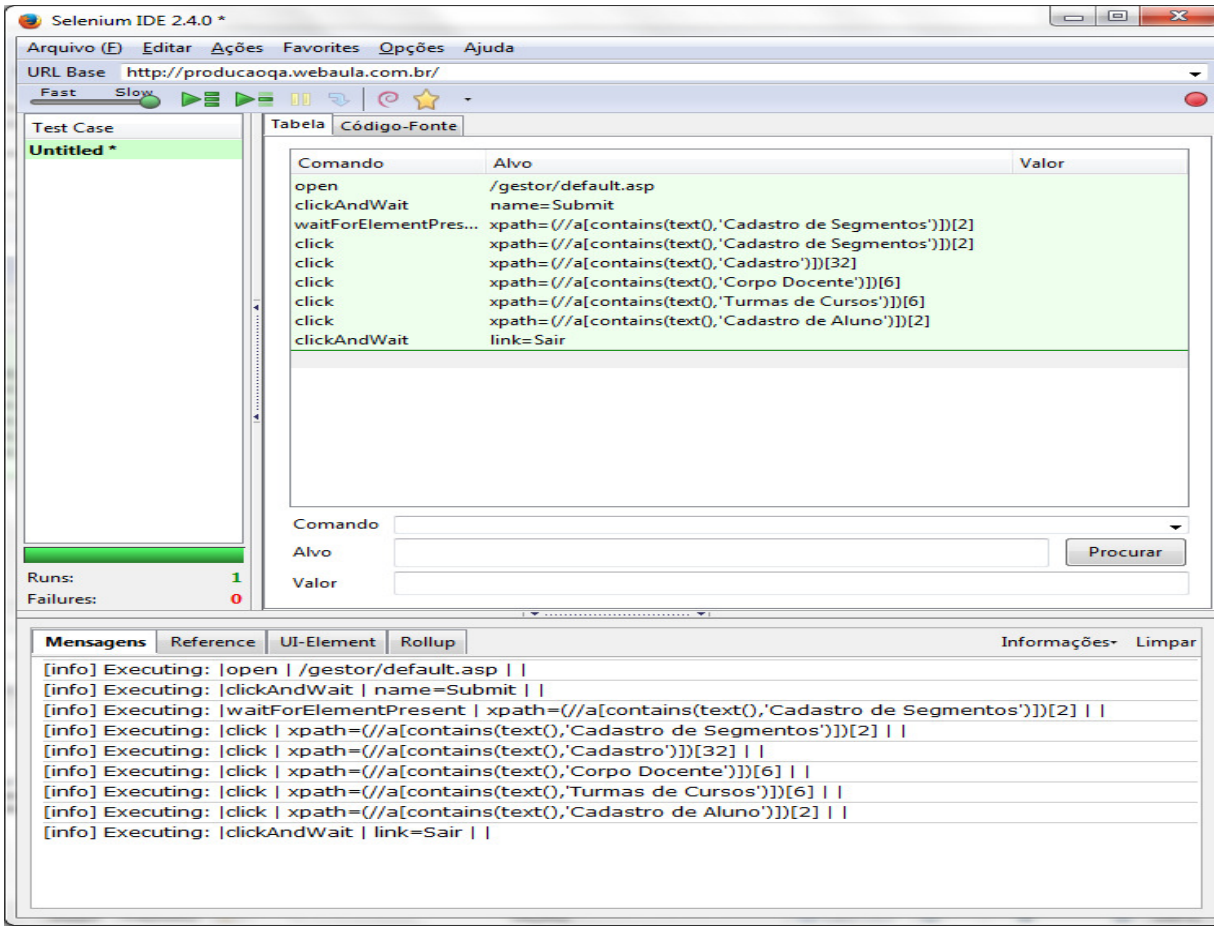
Tabela 01 – Execução Manual do *Smoke Test* e Regressão

Smoke Test	Regressão
13 minutos	4 horas e 40 minutos

Fonte: o autor

Para a automação do LMS, o QA utilizou primeiro o *Selenium IDE*. Conforme mostra a figura 06, o *IDE* capturou a navegação realizada pelo testador com os *clicks* do *mouse* e quando o *script* foi executado pelo testador, não ocorreram falhas ou erros.

Figura 06: Gravação com o *Selenium IDE*



Fonte: o autor

Em seguida, os *scripts* do *Selenium IDE* foram exportados para o *Selenium RC*. Após a exportação do código para a linguagem Java, a equipe utilizou o *eclipse* para melhorar os códigos gerados. A inserção de funções aumenta a qualidade do código, reduzem seu tamanho e proporcionam um melhor entendimento do *script* quando forem necessárias manutenções. O retrabalho também é minimizado, já que estas funções podem ser utilizadas em *scripts* futuros, descartando a necessidade de nova gravação pelo *IDE*. As imagens do projeto de automação de testes da *Web*Aula foram inseridas no "Anexo C".

Os *scripts* de automação de testes sempre inseriam os mesmos dados no sistema, independente se o testador precisaria de todos os dados para execução dos testes ou não. A

execução dos *scripts* de testes automatizados também foi cronometrada para que pudessem ser comparados, quanto ao tempo de execução, com os testes manuais, conforme descritos na tabela 02:

Tabela 02 – Execução Automatizada do *Smoke Test* e Regressão

Smoke Test	Regressão
18 minutos	5 horas e 03 minutos

Fonte: o autor

Desde que os testes de regressão foram iniciados pelo setor de qualidade, foram abertos vários chamados (SAC) solicitando correção ou melhoria no LMS, conforme exibido na tabela 03.

Tabela 03: Total de SACs antes do processo de automação

Recurso	Qtd. SACs Atendidas	Qtd. SACs em Aberto	SACs Melhoria	Qtd. Novas SACs	Total Geral
WebTv	0	17	0	0	17
Relatórios	1	14	1	0	16
Trabalhos	0	13	0	0	13
Cadastro de Usuario	1	11	1	0	13
Fórum	0	12	0	0	12
Turmas de Cursos	2	6	3	0	11
Comunidade	0	11	0	0	11
Cursos	1	8	1	1	11
Equipes e Filiais	0	9	0	0	9
Programa	0	2	5	2	9
Matrícula	0	7	0	2	9
Avaliação	0	5	3	0	8
Newsletter	0	5	3	0	8
Biblioteca Virtual	0	7	1	0	8
Chat	0	8	0	0	8
Pré-Inscrição	0	8	0	0	8
Agentes Inteligentes	0	2	2	2	6
Programa de Pontuação	0	6	0	0	6
Cartão LMS	0	3	0	2	5
Central de Mensagem	0	5	0	0	5
Expectativa*	0	5	0	0	5
Plano de Tutoria	0	5	0	0	5
Feedback	0	4	0	0	4
Itens de ajuda*	0	4	0	0	4
Login	0	3	1	0	4
Sala de Aula	0	4	0	0	4
Segmentos	0	3	1	0	4
Certificado	0	3	0	0	3
Banner*	0	1	0	0	1
Evento Conference*	0	1	0	0	1
Histórico	0	1	0	0	1
Termo de Uso*	0	1	0	0	1
Total Geral	5	194	22	9	230

Fonte: o autor

* Estes itens ainda não possuem especialistas, entretanto, entram no somário de chamados abertos pelo QA.

Entretanto, o tempo hábil para realizar a regressão completa no sistema foi reduzido, devido a outras demandas prioritárias ou até mesmo redução em dias entre a aplicação de um *build* e outro. Isso refletia negativamente, pois a cada *build*, poucos *bugs* eram encontrados pelo QA e acabavam por ser encontrados pelos clientes em produção.

Com a inserção da automação de testes, o setor iniciou testes mais detalhados no LMS afim de descobrir erros relacionados às regras de negócio do sistema e se especializar em suas principais funcionalidades. Foi feito um levantamento sobre quantos novos chamados foram abertos pela equipe nos principais recursos, conforme mostra a tabela 04.

Tabela 04 – Total de SACs após o processo de automação

Recurso	Qtd. SACs Atendidas	Qtd. SACs em Aberto	SACs Melhoria	Qtd. Novas SACs	Total Geral
Pré-Inscrição	1	7	0	17	25
Programa	0	4	5	14	23
WebTv	1	16	0	5	22
Turmas de Cursos	0	6	3	11	20
Relatórios	1	13	1	4	19
Fórum	0	12	0	6	18
Chat	0	8	0	10	18
Cursos	2	7	1	7	17
Biblioteca Virtual	0	7	1	8	16
Matrícula	0	9	0	6	15
Trabalhos	0	13	0	0	13
Cadastro de Usuario	0	11	1	0	12
Comunidade	0	11	0	0	11
Avaliação	0	5	3	3	11
Equipes e Filiais	1	8	0	1	10
Newsletter	0	5	3	0	8
Central de Mensagem	0	5	0	3	8
Sala de Aula	0	4	0	4	8
Agentes Inteligentes	0	4	2	0	6
Programa de Pontuação	0	6	0	0	6
Cartão LMS	0	5	0	0	5
Expectativa*	0	5	0	0	5
Plano de Tutoria	0	5	0	0	5
Feedback	0	4	0	0	4
Itens de ajuda*	0	4	0	0	4
Login	0	3	1	0	4
Segmentos	0	3	1	0	4
Certificado	0	3	0	0	3
Banner*	0	1	0	0	1
Evento Conference*	0	1	0	0	1
Histórico	0	1	0	0	1
Termo de Uso*	0	1	0	0	1
Total Geral	6	197	22	99	324

Fonte: o autor

* Estes itens ainda não possuem especialistas, entretanto, entram no somário de chamados abertos pelo QA.

4.2 Análise dos dados

Conforme descrito no capítulo anterior, a análise levou em consideração o pior e o melhor cenários já vivenciados pelos testadores, *build* aplicado todos os dias ou a cada 15 dias, respectivamente. No LMS, alguns cadastros dependem de outros, sendo necessária uma sequência de passos. O testador que realizou a execução dos testes manuais conhecia o sistema e seguiu esta sequência realizando os mesmos cadastros que constam no *script*, se equiparando aos testes automatizados. A escolha de um testador que possui pouco conhecimento do sistema poderia elevar o tempo estimado para execução dos testes, alterando o resultado da análise.

Após o levantamento de dados realizado anteriormente na empresa *Web*Aula, foi constatado que as execuções dos testes manuais foram realizadas um pouco mais rápidas que a dos testes automatizados. Isso ocorre porque nos *scripts*, antes de ser realizado qualquer cadastro, é feito a verificação se o dado a ser inserido já consta no sistema e em alguns casos, o dado era excluído e em seguida cadastrado. Esta verificação era necessária porque em determinados testes, para não alterar o resultado esperado do teste, o dado a ser inserido não podia ter ações anteriormente realizadas.

Todos os *scripts* do projeto possuem vários métodos em sua estrutura, e normalmente, o último método exclui todos os dados inseridos no início do *script*. Entretanto, se por algum motivo ocorresse um erro no *script*, por exemplo, e ele não fosse executado até o fim, os dados não seriam excluídos. Como os nomes dos dados inseridos pelos *scripts* não são alterados, havia o risco de algum dado ainda estar cadastrado. Ao realizar testes manualmente, o testador não realizava esta verificação, ele já inseria dados novos para os testes.

A criação dos *scripts* foi um processo demorado na empresa *Web*Aula. Os profissionais não conheciam a ferramenta que estavam utilizando, e precisaram aprender como usá-la. A gravação dos *scripts* pelo *Selenium IDE* era rápida, entretanto, a execução às vezes era bem demorada. Por se tratar de um sistema *web*, quando a navegação ficava lenta, ocorria erro na execução, sendo preciso tratar os *scripts* para tais imprevistos que poderiam acontecer.

E ao converter para o *Selenium RC*, o que estava executando no *IDE* já não executava, sendo preciso horas e em alguns casos mais complexos, dias, para colocar um *script*

funcionando. Entretanto, após criados e em funcionamento, trouxeram diversos benefícios para a *Web*Aula.

Com os dados levantados quanto à execução dos testes manuais, foi possível mensurar em curto, médio e longo prazo o tempo que cada testador da equipe passou a ter para realização de atividades que necessitam ser executadas manualmente, conforme mostra a tabela 05. Quando a análise foi iniciada a equipe possuía 6 testadores e durante a análise foram contratados mais 3 profissionais.

Tabela 05 – Tempo destinado por testador para *Smoke Test* e Regressão:

Atividade Realizada	Aplicação do *Build* em Homologação	Tempo Destinado para a tarefa	Em 1 mês*	Em 6 meses*	Em 1 ano*
Smoke Test	Diariamente	13 minutos	4 horas e 33 minutos	26 horas	52 horas
	A cada 15 dias	13 minutos	26 minutos	3 horas	5 horas e 20 minutos
Regressão	A cada 15 dias	4 horas e 40 minutos	9 horas e 20 minutos	56 horas	112 horas

Fonte: o autor

*O calculo realizado considera 20 dias uteis por mês.

Considerando que o tempo que passou a ser destinado a outras atividades era para 9 profissionais, obteve-se um valor considerável, que proporcionou a equipe de testes maior conhecimento do LMS e de outros tipos de testes e à empresa, aumento da qualidade do produto.

A *Web*Aula tinha uma deficiência quanto a especialistas em regras de negócio do sistema LMS, até mesmos os analistas de requisitos da empresa possuíam esta deficiência. Após a automação de testes no setor de qualidade da empresa, o QA se tornou uma referência para toda a empresa quanto a regras de negócio das principais funcionalidades do sistema.

Com a ampliação do conhecimento, foi possível identificar diversas falhas relacionadas a regras de negócio nas funcionalidades. Em aproximadamente dois meses de atuação, foram abertos noventa e nove novos chamados, e antes da automação, a cada quinze dias eram abertos em média nove, ou seja, no período analisado, com a automação de testes, foram abertos 63% de chamados a mais do que seriam abertos sem o auxilio dela.

A equipe de qualidade agora atua em melhorias propostas para o sistema desde seu nascimento. Junto com os analistas de requisitos, analisa as propostas e os orienta na criação dos requisitos funcionais, não funcionais e protótipos de tela, baseado nas regras de negócio do LMS. A chance de se cometer erros é minimizada antes mesmo da proposta ir para aprovação do cliente.

A automação de testes proporcionou ainda a ampliação dos testes no setor, quanto à implantação de testes de caixa branca, que até então, não eram realizados pela equipe. Como o teste maçante agora é executado pelos *scripts* automatizados, a equipe teve tempo para aprender e aplicar na prática os conhecimentos de testes estruturais no código fonte. Quando os testes são iniciados desde o desenvolvimento do código, os custos com correções são menores e a qualidade do produto aumenta significativamente, pois evita que erros como os de lógica e regras de negócio sejam inseridos no produto.

A cobertura de testes no sistema aumentou com a automação de testes, já que seu *script* sempre realiza os cadastros básicos do sistema, realizando o *smoke test* em suas principais funcionalidades, o que nem sempre era feito manualmente. Caso o *script* identificasse algum erro bloqueante durante a execução, o *build* era rejeitado antes mesmo de ser liberado para os testadores.

Para os testes de regressão também houve aumento na cobertura de testes, pois quando ocorria o pior cenário, o QA não teria tempo hábil de realizar os testes de regressão manualmente. Os automatizados entraram para suprir esta necessidade, assim que o *build* era homologado e aplicado em produção, os *scripts* eram executados. Mesmo que fosse liberado novo *build* para ser homologado no dia seguinte, a regressão já teria sido realizada pelos *scripts*.

Após a aplicação do *build* em produção, os *scripts* eram executados e muitos erros graves começaram a ser identificados por eles, permitindo sua correção imediata, evitando que o cliente os encontrasse em produção. Antes as correções ocorriam tardiamente, clientes encontravam diversos erros e dependendo da sua gravidade e do contrato do cliente, a *Web*Aula tinha um prazo para corrigir o problema, caso não cumprisse, pagava multa. Em alguns casos, devido à insatisfação do cliente, perdia o contrato.

5 CONCLUSÃO

A execução dos testes manuais foi realizada alguns minutos mais rápido que a dos *scripts* automatizados. Por se tratar de uma diferença de tempo tão pequena perto dos benefícios já alcançados pela empresa, se considera viável a automação de testes de regressão e *smoke test*, já que estes são considerados maçantes e não necessitam de raciocínio para a execução.

Quando realizados manualmente, estes testes eram divididos e realizados por todos os membros da equipe, que encaixadas entre outras atividades, demoravam até dias para serem concluídas. E nem sempre eram concluídas pelos testadores, já que poderiam surgir atividades prioritárias e/ou até mesmo aplicação de *build* antes da data prevista, abortando os testes de regressão.

Os *scripts* automatizados poderiam ser executados a qualquer dia, a qualquer hora e quantas vezes fosse necessário. Apenas um profissional seria desviado de suas funções para executá-los e esta execução seria concluída em algumas horas. Com a execução dos testes automatizados, o QA muitas vezes, se antecipa ao cliente, ao encontrar antes deles *bugs* que são corrigidos antes de tomarem conhecimento de sua existência.

O tempo que os profissionais destinavam aos testes rotineiros de regressão e *smoke test*, agora é destinado a atividades que necessitam de execução manual. Os integrantes da equipe agora são especialistas em regras de negócios das principais funcionalidades do sistema LMS. Estas foram divididas entre eles e cada um aprofundou os testes nas funcionalidades de sua responsabilidade, podendo assim auxiliar outras equipes quanto à análise e desenvolvimento de melhorias.

A equipe introduziu em sua rotina, a realização de testes de caixa branca, aumentando o escopo de testes no sistema. O QA realizava apenas testes de caixa preta no sistema LMS, pois não tinha tempo hábil para introduzir outros tipos de testes durante o *build* em homologação. Com a automação de testes, o tempo a mais disponível de cada integrante aumentou, possibilitando a inserção do projeto de testes de caixa branca a cada *build* em homologação.

Com o setor de qualidade mais qualificado, conhecendo melhor o produto e suas regras de negócio, a inserção de novos tipos de testes no produto e a cobertura de testes que os automatizados veem realizando, a qualidade do LMS está crescendo cada vez mais.

A automação de testes é um complemento aos testes manuais, auxiliando os testadores quanto à execução de alguns testes. Quando realizados em conjunto, agregam mais qualidade ao produto e reduzem custos na empresa em longo prazo. Além disso, permite ao profissional do setor de qualidade ganho de conhecimento no sistema e em outros tipos de teste.

Pode-se concluir que o objetivo desta monografia, de apontar os benefícios que a *Web*Aula já alcançou desde a implantação da automação de testes no setor de qualidade da empresa, foi atingida.

Neste trabalho não foi abordada a automação de outros tipos de testes, como os de performance, carga e *stress*, já que estes tipos de testes ainda não foram introduzidos no setor de qualidade da empresa *Web*Aula. Entretanto, recomenda-se em trabalhos futuros esta abordagem, a fim de levantar os pontos positivos e negativos em relação aos mesmos.

REFERÊNCIAS BIBLIOGRÁFICAS

BARTIÉ, Alexandre. **Garantia da qualidade de software: adquirindo maturidade organizacional.** Rio de Janeiro: Campus, 2002. 291 p.

BASTOS, Anderson et al. **Base de conhecimento em teste de software**. São Paulo: Martins, 2007.

CAETANO, Cristiano. **Conhecendo os testes exploratórios.** Disponível em: < http://www.devmedia.com.br/space.asp?id=195255> Acesso em 27 mar. 2013.

CAETANO, Cristiano. **Automação e Gerenciamento de Testes: Aumentando a Produtividade com as Principais Soluções Open Source e Gratuitas** (2a edição). Disponível em: <http://www.linhadecodigo.com.br/artigo/1566/automacao-e-gerenciamento-de-testes-aumentando-a-produtividade-com-as-principais-solucoes-open-source-e-gratuitas-2a-edicao.aspx>. Acesso em 27 mar. 2013.

CAMPOS, Augusto. **O que é software livre**. BR-Linux. Florianópolis, março de 2006. Disponível em < http://br-linux.org/2008/01/faq-softwarelivre.html>. Acesso em 09 set. 2013.

CAMPOS, Fábio Martinho. **Qualidade, Qualidade de Software e Garantia da Qualidade de Software são as mesmas coisas?**. Disponível em: <http://www.linhadecodigo.com.br/artigo/1712/qualidade-qualidade-de-software-e-garantia-da-qualidade-de-software-sao-as-mesmas-coisas.aspx>. Acesso em 05 mai. 2013.

Definição de Código Aberto. Disponível em < http://opensource.org/osd >. Acesso em 11 nov. 2013.

FILHO, Wilson De Pádua Paula. **Alguns Fundamentos da Engenharia de Software.** Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-alguns-fundamentos-da-engenharia-de-software/8029#>. Acesso em 02 mai. 2013.

HETZEL, W., **Guia completo ao teste de software**. Rio de Janeiro: Campus, 1987.

INTHURN, Cândida. **Qualidade & teste de software**. Florianópolis. Visual Books, 2001.

ISTQB WG Foundation Level, Thomas Muller (chair), Debra Friedenberg. **Certified Tester Foundation Level Syllabus**. Tradução realizada pela TAG01 - Documentação do BSTQB baseada na versão 2011.

MASCARENHAS, Sidney Augusto. **Metodologia Científica**. São Paulo: Pearson Education do Brasil, 2012.

MCCONNELL, Steve, **Rapid Development: Taming Wild Software Schedules,** Microsoft Press, 1996, p. 73, 74

MOLINARI, Leonardo. **Testes Funcionais de Software**. Florianópolis: Visual Books, p.48-128, 2008.

NETO, Arilo Claudio Dias. **Introdução a Teste de Software**. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em 10 mai. 2013.

PRESSMAN, R. S., **Software engineering: A practitioner's approach**. 4th. ed. McGraw-Hill, 1997. p. 22-53.

PRESSMAN, R. S., **Software Engineering: A Practitioner's Approach**. McGraw-Hill, 6th ed, Nova York, NY, 2005.

PRESSMAN, Roger. **Engenharia de software**. 2006. McGraw Hill. Capítulo 2 – Processo: Uma visão Genérica;

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. Rio de Janeiro: McGraw-Hill, 2006.

ROCHA, A. R. C., 1987, **Análise e Projeto Estruturado de Sistemas**, Editora Campus, Rio de Janeiro.

SCHWARTZ, J. I. **Construction of software. In: Practical Strategies for Developing Large Systems**. Menlo Park: Addison-Wesley, 1st. ed., 1975.

SOMMERVILLE, I. **Software engineering**. 5th. ed. Addison-Wesley, 1995. p. 7.

SOMMERVILLE. **Software Engineering**. 6thEdition, AddisonWesley, 2000.

SOMMERVILLE, Ian. **Engenharia de Software**, 8ª edição. Tradução: Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edílson de Andrade Barbosa; revisão técnica Kechi Hirama. – 8.ed.- São Paulo: Pearson Addison Wesley, 2007.

THAMIEL, Thiago. **Ciclo de Vida do Scrum**. Disponível em: <http://thiagothamiel.com/tag/scrum/page/2/ >. Acesso em: 09 set. 2013.

ANEXO A – Entrevista com coordenador de qualidade da *web*Aula

Idade: 40 anos
Sexo: Masculino

O que desencadeou a necessidade da inclusão da automação de testes no setor de qualidade da empresa *Web*Aula?
Os trabalhos desenvolvidos pela equipe de qualidade tinham como foco os testes funcionais somente. O escopo abrangido por este tipo de teste, bem como sua efetividade, não atendia completamente às necessidades da empresa. Ainda tínhamos problemas básicos sendo enviados para os ambientes de produção. Isso tinha que ser resolvido.

Quais resultados eram esperados com a automação de testes?
O foco do projeto de automação, adotado inicialmente, tentava resolver problemas como: Tempo gasto na realização dos testes; entrega das demandas para os clientes dentro de prazos compromissados; desempenho do profissional de testes mais especializado nas regras de negócios e menos na parte funcional dos recursos e uma expectativa de retorno financeiro para a empresa por menos homens/hora para cada demanda, a médio e longo prazo.

Com o projeto em andamento, quais destes resultados já foram alcançados?
Já obtivemos um retorno muito positivo quanto ao tempo gasto na realização de teste sobre regras de negócio, além da melhora no filtro das falhas básicas e primárias, algumas vezes não identificadas no dia a dia, que seguem de alguma forma para produção.

O setor de qualidade da empresa conseguiu aumentar a cobertura dos testes e consequentemente a qualidade do produto?
Sim, a divisão dos produtos em recursos e estes, por sua vez, em requisitos, permitiu à equipe de testes uma especialização mais próxima às regras de negócios, envolvida em cada um deles. Assim, a qualidade aumentou à medida que se conhece mais o produto e como ele funciona. Os testes automatizados permitiram um refinamento na parte funcional de cada um destes recursos, liberando o *tester* para melhorar a qualidade e aumentar o escopo atingido.

Quais os pontos positivos e os pontos negativos foram identificados após a inclusão do projeto?

Bem, os pontos positivos, como relatados, são mais perceptíveis diretamente quando avaliamos o tempo de testes e a cobertura sobre as regras de negócios. Já para os pontos negativos, acredito que as ferramentas utilizadas exigem conhecimentos de outros nichos de TI, como programação, linguagem e logica, o que nem sempre são os principais pontos desenvolvidos nos profissionais de testes. Também vejo o tempo de implementação dos *scripts* devido à linguagem utilizada nos produtos legados da empresa.

Como os gestores da *Web*Aula veem o projeto de automação de testes na empresa?

A *Web*Aula entende a importância de se ter um produto entregue com a máxima qualidade, em menor tempo e que satisfaça ao cliente. Sendo assim, a iniciativa de automatizar parte do esforço de testes foi aceita como uma grande promessa para atingir a sua filosofia. No seu mercado de atuação, o tempo é fundamental, e algumas vezes decisivo, na conquista ou perda de um cliente e os testes automatizados seguem na linha da politica adotada na *Web*Aula: de que o tempo é dinheiro.

ANEXO B – Principais telas do LMS

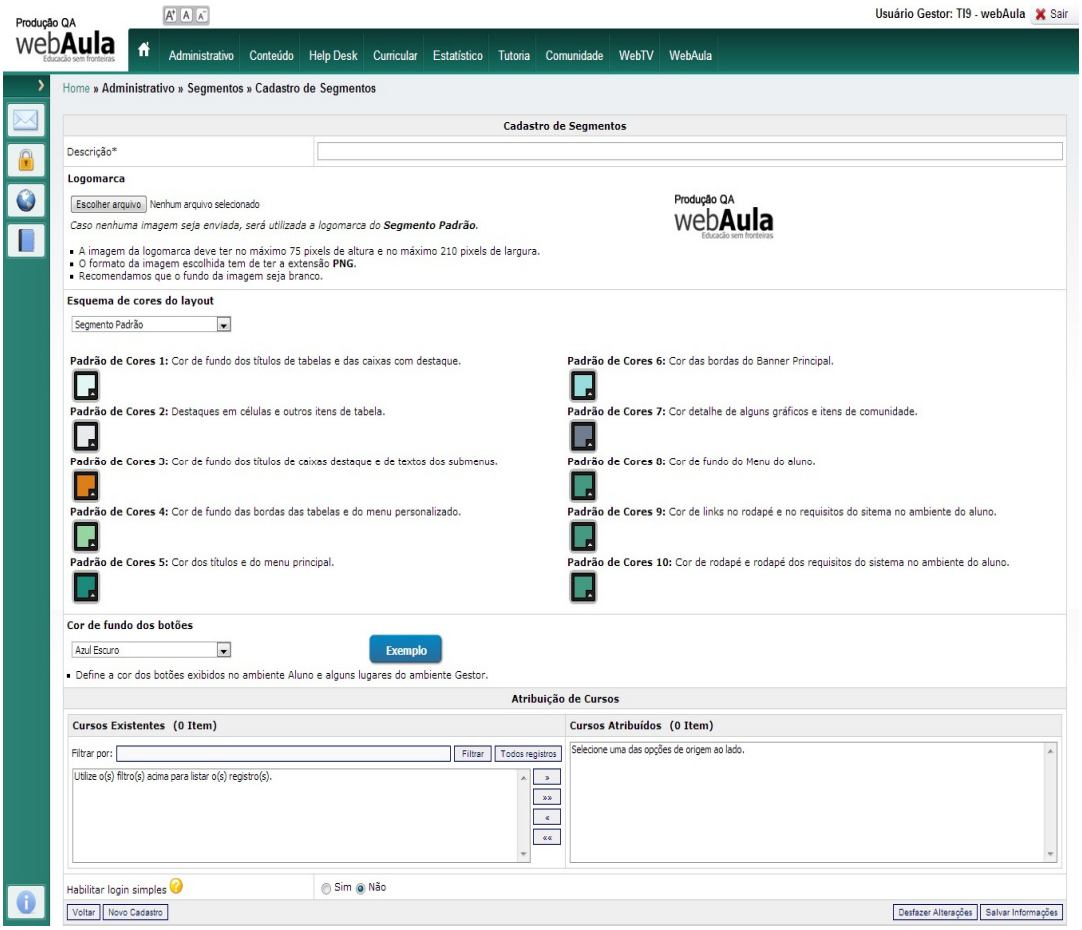
Na figura 07 é exibida a tela de *login* do gestor do LMS, onde o usuário deverá inserir *login* e senha para acessar o sistema.

Figura 07 – Tela de *Login* Gestor

Fonte: o autor

O gestor poderá definir um ou mais segmentos para seus alunos. Na figura 08 é exibida a tela para cadastro destes segmentos. O segmento permite que determinados recursos fiquem disponíveis apenas para os alunos dos segmentos selecionados.

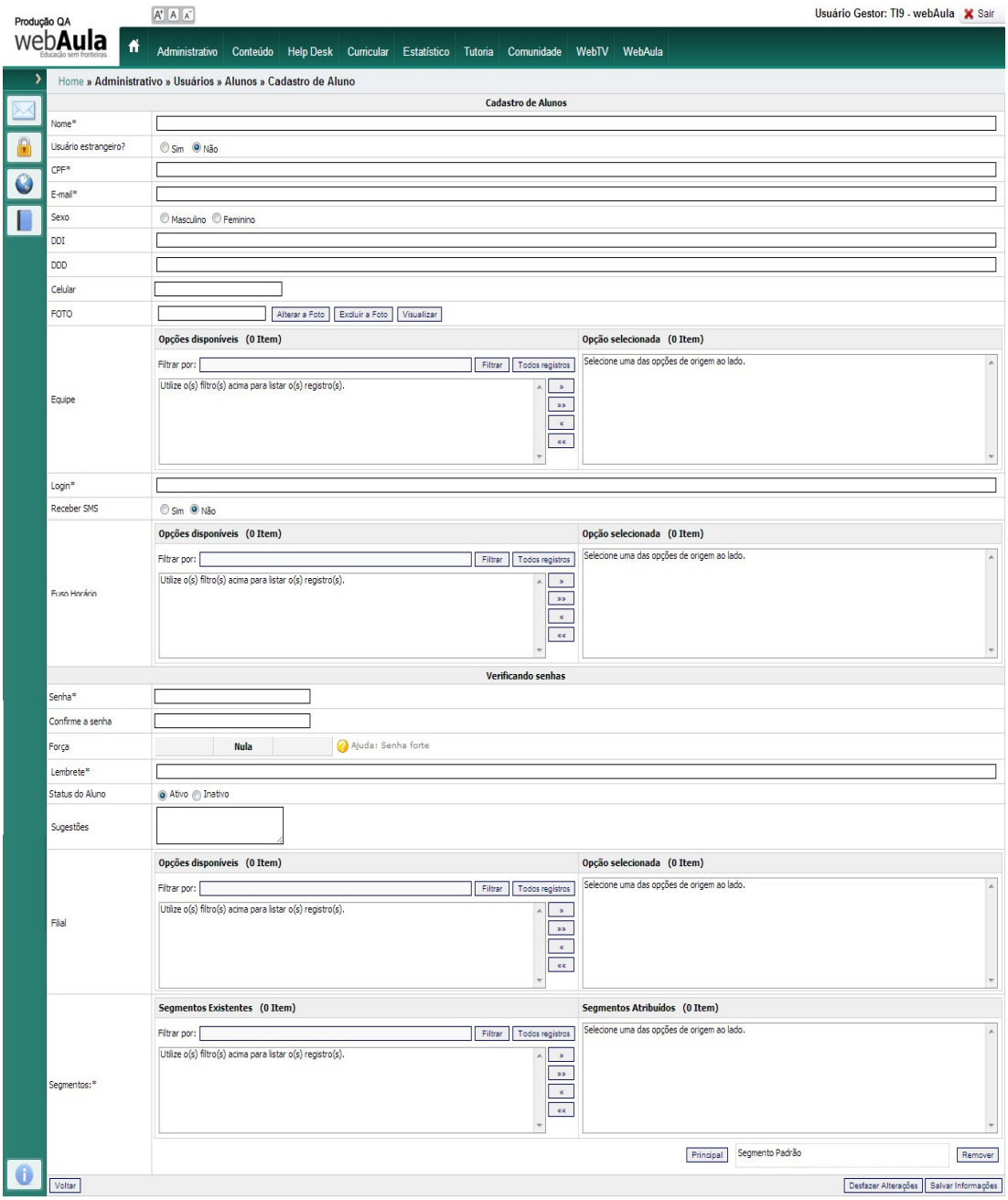
Figura 08 – Tela para Cadastro de Segmento



Fonte: o autor

A figura 09 exibe a tela de cadastro de alunos, onde o gestor poderá inserir manualmente alunos no sistema. Os alunos cadastrados poderão realizar os cursos e acessar os demais recursos da plataforma de acordo com as definições do gestor.

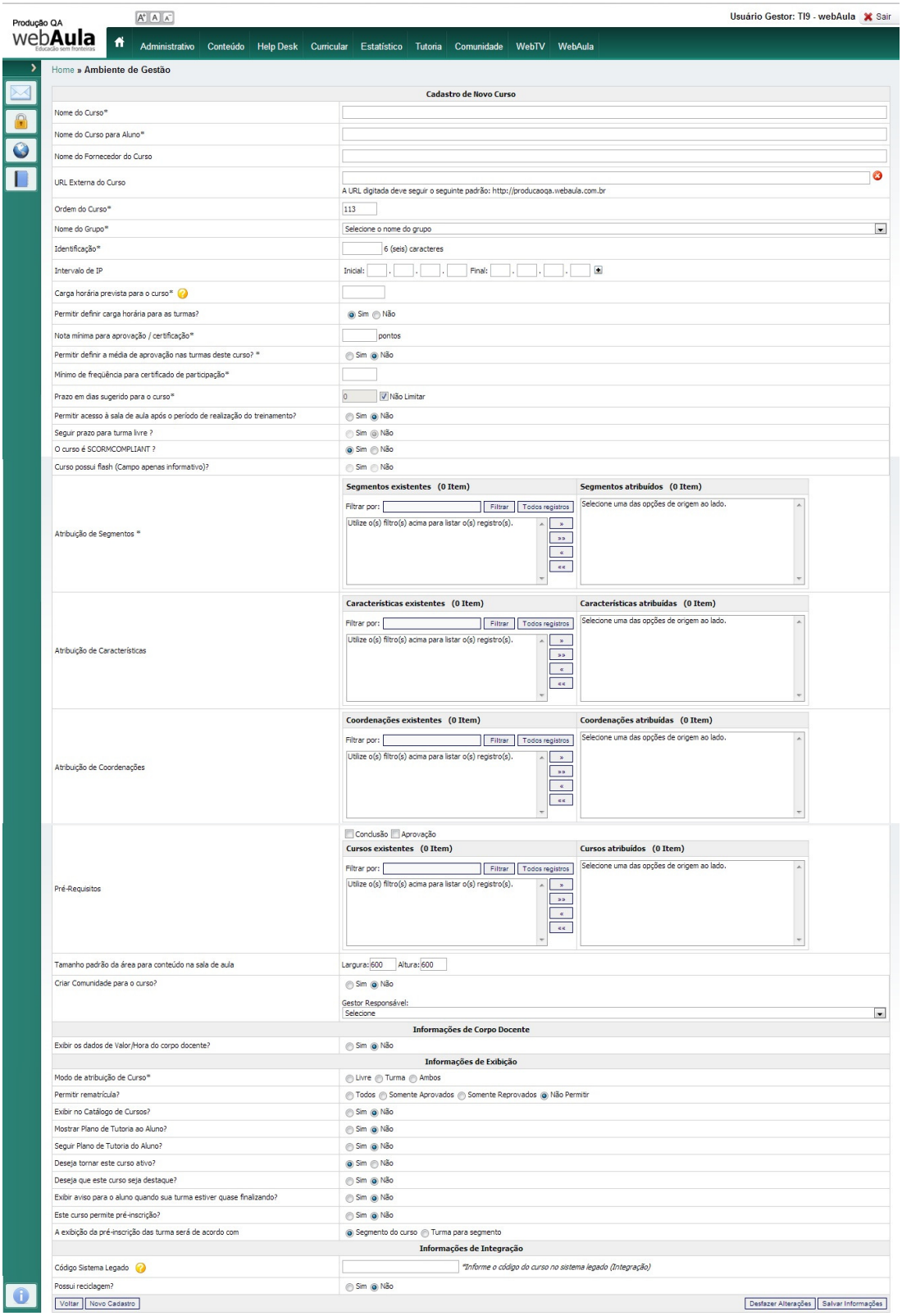
Figura 09 – Tela de Cadastro de Alunos



Fonte: o autor

No cadastro de curso o gestor pode definir carga horária do curso, notas mínimas para aprovação/certificação dentre outras configurações conforme pode ser visto na figura 10.

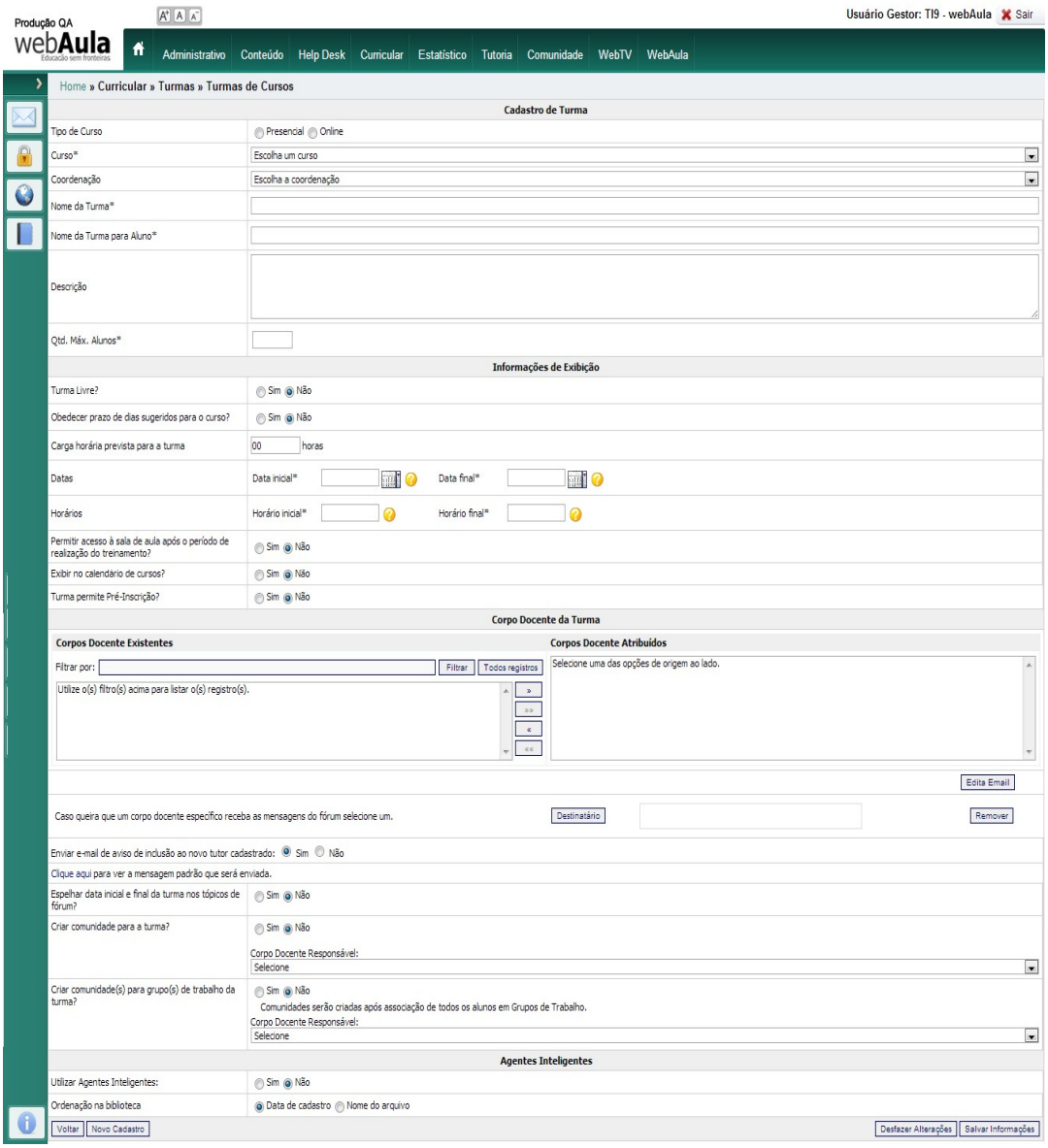
Figura 10 – Tela de Cadastro de Curso *Online*



Fonte: o autor

Para cada curso cadastrado, o gestor poderá inserir quantas turmas desejar. A figura 11 mostra diversas configurações ao cadastrar uma turma, muitas delas são realizadas por turma, ou seja, em um mesmo curso pode haver, por exemplo, turmas com datas e horários distintos.

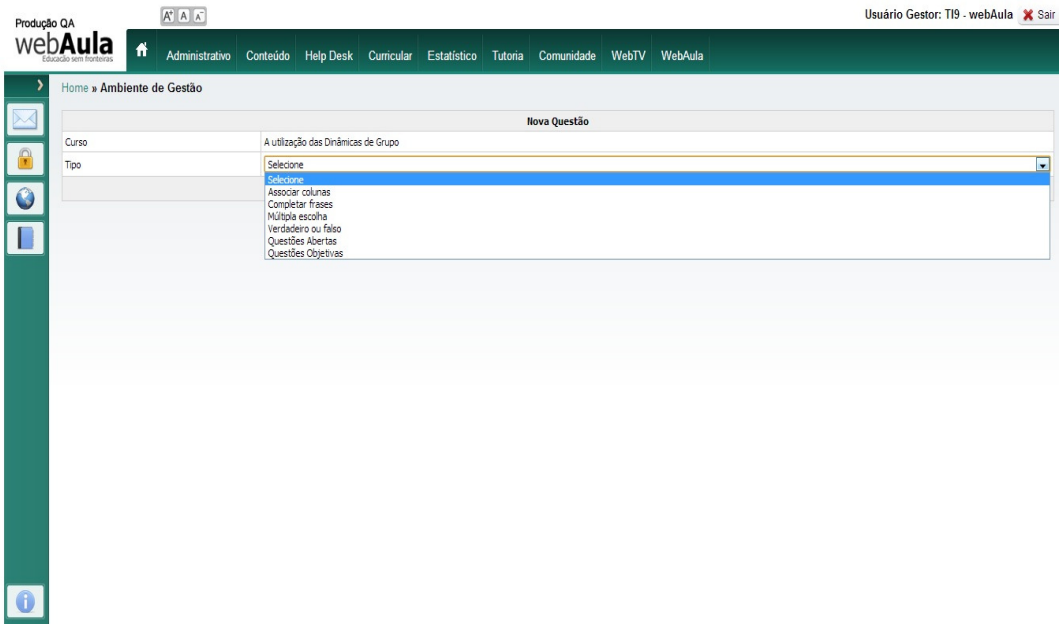
Figura 11 – Tela de Cadastro de Turmas



Fonte: o autor

É possível cadastrar banco de questões para as avaliações e exercícios dos cursos. Conforme pode ser visto na figura 12, o sistema permite o cadastro de questões dos tipos: associar colunas, completar frases, múltipla escolha, verdadeiro ou falso, questões abertas e questões objetivas.

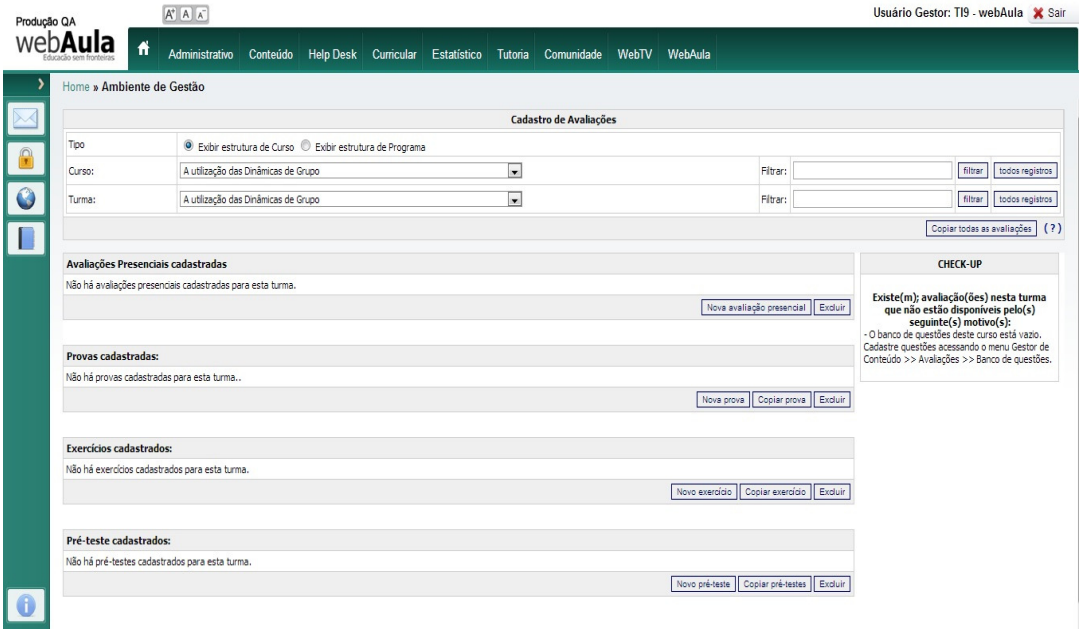
Figura 12 – Tela de Cadastro de Banco de Questões



Fonte: o autor

Após o cadastro das questões, o gestor cadastra as avaliações, pré-testes e exercícios para cada turma, conforme pode ser visto na figura 13.

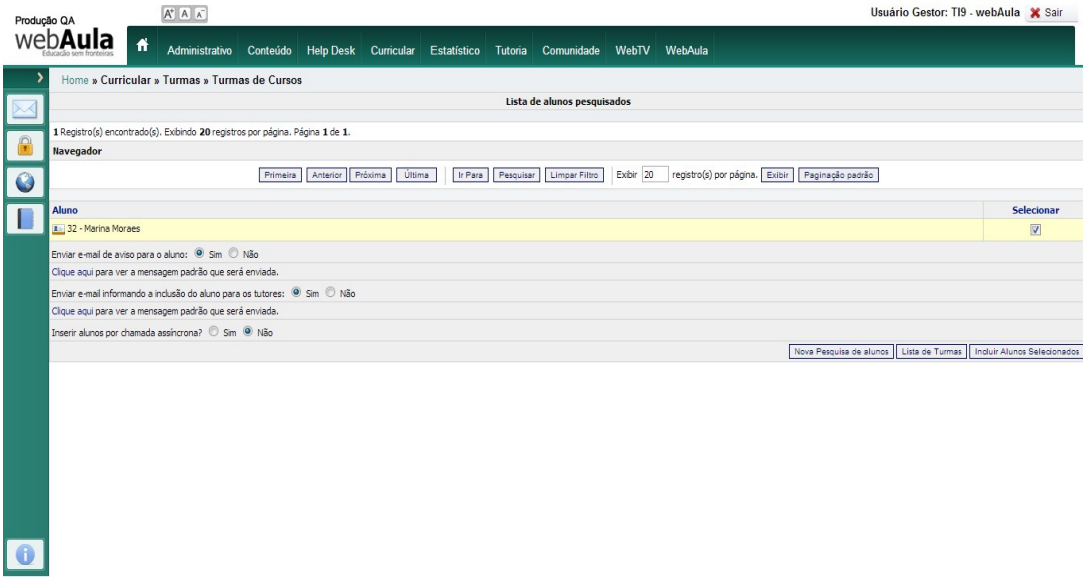
Figura 13 – Tela de Cadastro de Avaliações



Fonte: o autor

A figura 14 mostra a inclusão de um aluno em uma turma de curso pelo gestor. O aluno matriculado terá acesso a todos os recursos configurados na turma de curso.

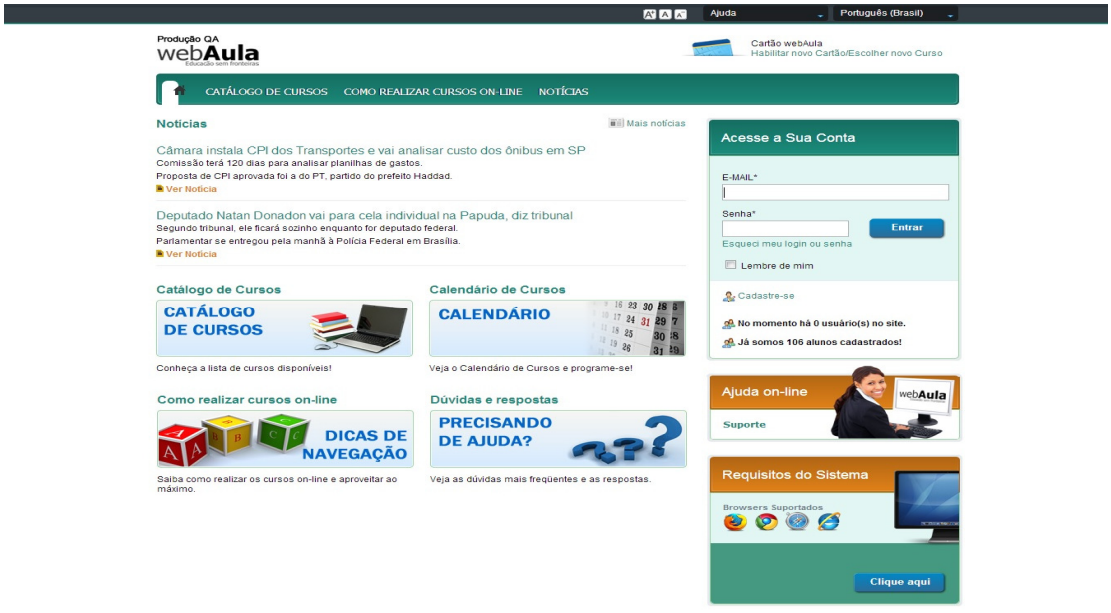
Figura 14 – Tela de Matrícula de Aluno em Turma de Curso *Online*



Fonte: o autor

Já no ambiente aluno, conforme mostra à figura 15, o aluno deverá inserir *e-mail* e senha para ter acesso ao sistema.

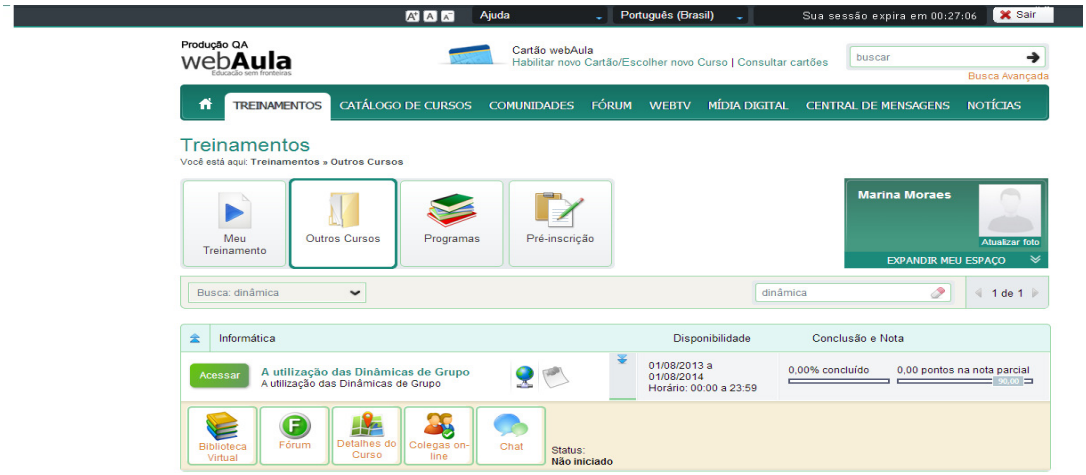
Figura 15 – Tela de *Login* Aluno



Fonte: o autor

A figura 16 mostra a visão do aluno a seu curso e aos recursos disponíveis para aquele curso, bem como disponibilidade do curso, percentual de conclusão do curso e nota obtida em avaliações. Através desta tela o aluno poderá acessar a sala de aula para realizar o curso.

Figura 16 – Tela Outros Cursos



Fonte: o autor

O aluno poderá acessar a sala de aula do curso ao qual pertence, conforme exemplificado na figura 17.

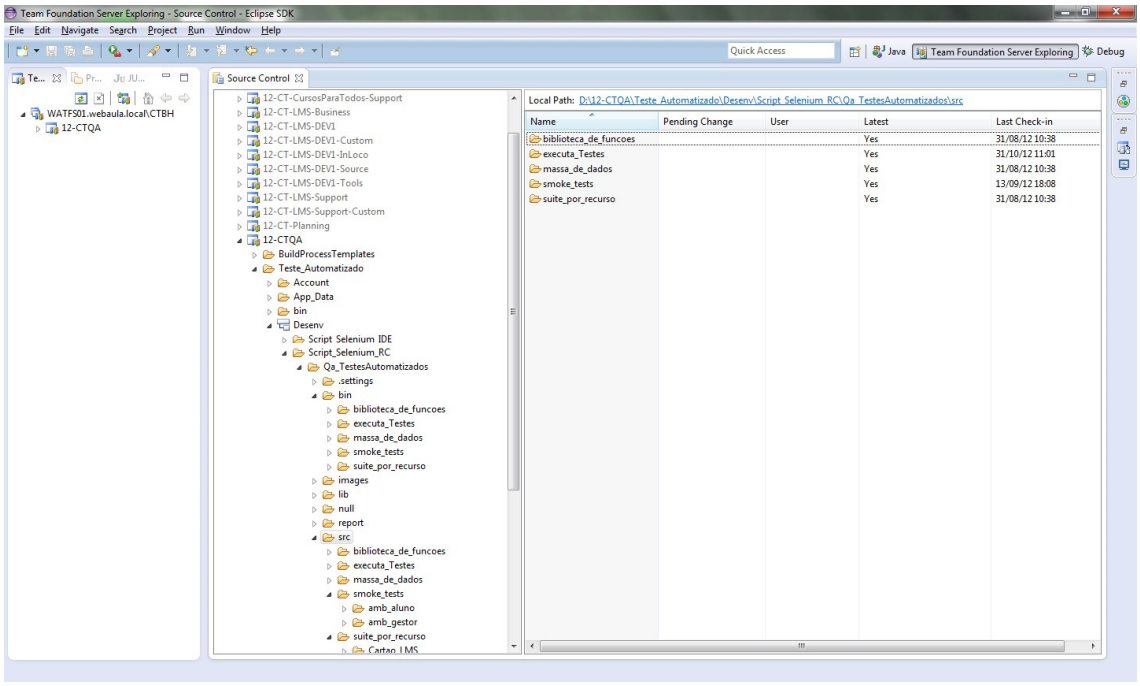
Figura 17 – Sala de Aula Virtual

Fonte: o autor

ANEXO C – Projeto de automação de testes

Na figura 18, é exibida a estrutura do projeto de automação de testes da empresa *Web*Aula.

Figura 18 – Estrutura do Projeto de Automação de Testes



Fonte: o autor

Vários *scripts* de testes já foram criados desde sua implantação. Abaixo, serão listados alguns dos *scripts* existentes.

A figura 19 mostra uma função criada pela equipe, *waitForElementPresent*, sempre que for utilizada, o sistema irá esperar 30 segundos para que o elemento solicitado seja exibido na tela.

Figura 19 – Funções Complementares

Source Control | Funcoes_Complementares.java

```
1  package biblioteca_de_funcoes;
2
3
4  import com.thoughtworks.selenium.*;
7
8
9  public class Funcoes_Complementares extends SeleneseTestCase{
10
11     private Selenium selenium;
12
13     //construtor
14     public Funcoes_Complementares(Selenium browser){
15
16         this.selenium = browser;
17     }
18
19
20  /*Função para localizar elementos HTMl na tela.
21     EX: ID=Submit link=textoDolink
22   */
23     @SuppressWarnings("deprecation")
24     public void waitForElementPresent(String elemento)
25     {
26         //espera por 30 segundos para trazer algum elemento
27         pause(2000);
28         int x=0;
29         while(x<30)
30         {
31             pause(1000);
32             if(selenium.isElementPresent(elemento))
33                 break;
34             else
35             {
36                 /*if(x==29)
37                     fail("timeout");*/
38                 x++;
39                 continue;
40             }
41         }
42     }
43
44     @SuppressWarnings("deprecation")
45     public void waitForTextPresent(String texto)
46     {
47         for (int second = 0;; second++) {
48             if (second >= 90) fail("timeout");
49             try { if (selenium.isTextPresent(texto)) break; } catch (Exception e) {}
50             pause(1000);
51         }
52     }
53
```

Fonte: o autor

Para a realização de qualquer ação no sistema, o gestor precisaria entrar com *login* e senha, então, para cada *script* criado no ambiente gestor, *login* e senha deveriam ser gravados. Para agilizar o processo, a equipe criou funções para as funcionalidades que seriam usadas mais de uma vez, entre elas, LogarAmbienteGestor, conforme é mostrado na figura 20.

Figura 20 – Funções LMS

Source Control | Funcoes_LMS.java

```
1  package biblioteca_de_funcoes;
2
3  import org.junit.Test;
6
7
8  public class Funcoes_LMS extends SeleneseTestCase
9  {
10
11             private Selenium selenium;
12             private static Funcoes_Complementares f_comp;
13             private static int c=0;
14
15
16
17             //constrcutor
18             public Funcoes_LMS(Selenium browser)
19             {
20                 this.selenium = browser;
21             }
22
23
24             public void logarAmbienteGestor(String login, String senha)
25             {
26                 f_comp = new Funcoes_Complementares(selenium);
27
28                 selenium.windowMaximize();
29
30                 selenium.open("/gestor/frames.asp");
31                 f_comp.waitForElementPresent("id=txtlogin");
32                 selenium.type("id=txtlogin",login);
33                 selenium.type("id=txtsenhas",senha);
34                 selenium.click("name=Submit");
35                 f_comp.waitForElementPresent("css=#localizacao > b");
36
37                 if(login.toLowerCase()=="ti9")
38                     configuraParametros();
39             }
40
41             @SuppressWarnings("deprecation")
42             public void logarAmbienteAluno(String login, String senha)
43             {
44                 f_comp = new Funcoes_Complementares(selenium);
45
46                 selenium.windowMaximize();
47
48                 selenium.open("/aluno/login/");
49                 f_comp.waitForElementPresent("id=txtUsuario");
50                 selenium.type("id=txtUsuario",login);
51                 selenium.type("id=txtSenha",senha);
52                 selenium.click("id=btnEnviar");
53                 pause(8000);
54
55                 if(selenium.isElementPresent("id=txtEmail"))//janela "Atenção"
56                 {
57                     selenium.click("name=submit");
58                 }
59
60
61                 if(selenium.isElementPresent("name=CPF"))//janela "Meu Cadastro"
62                 {
63                         assertTrue(selenium.isTextPresent("Meu Cadastro"));
64
65                         try{
66                             selenium.selectWindow("Produção QA");
67                             }
68                         catch(Exception e){}
69
70                         pause(1000);
71                         selenium.click("id=btnSalvarInformacoes");
72                         f_comp.waitForElementPresent("id=tdMensagemCaixaMensagem");
73                         assertEquals("Informações armazenadas com sucesso.", selenium.getText("id=tdMensagemCaixaMensagem"));
74
75                         try{
76                             selenium.selectWindow("Produção QA");
77                             }
78                         catch(Exception e){}
79                         pause(100);
80                         f_comp.waitForElementPresent("link=Fechar");
81                         selenium.click("link=Fechar");
82                         pause(500);
83                 }
84                 pause(4000);
85                 try{
86                     if(selenium.isElementPresent("link=Treinamentos"))
87                     {
88                         try{
89                             selenium.selectWindow("Produção QA");
90                             }
91                         catch(Exception e){}
92                         pause(1000);
93                         selenium.click("link=Treinamentos");
94                         pause(4000);
95                     }
96                 }
97                 catch (Exception e){}
98             }
99
100
```

Fonte: o autor

Os scripts criados para *smoke test* dos ambientes gestor e aluno podem ser vistos na figura 21.

Figura 21 – *Smoke Test* Gestor e Aluno

Source Control | Executa_Smoke_Tests.java

```
3  package executa_Testes;
4
5  import junit.framework.Test;
9
10
11 public class Executa_Smoke_Tests
12 {
13     public static Test suite() {
14         TestSuite suite = new TestSuite();
15
16         //Smoke tests Ambiente Gestor
17         suite.addTestSuite(Cad_Alunos.class);
18         suite.addTestSuite(Blq_Lib_Aluno.class);
19         suite.addTestSuite(Cad_Avaliacoes.class);
20         suite.addTestSuite(Cad_Banco.class);
21         suite.addTestSuite(Cad_Caracteristicas.class);
22         suite.addTestSuite(Cad_Cargo.class);
23         suite.addTestSuite(Cad_Clipping.class);
24         suite.addTestSuite(Cad_Contato.class);
25         suite.addTestSuite(Cad_Coordenacao.class);
26         suite.addTestSuite(Cad_Dicas.class);
27         suite.addTestSuite(Cad_DicionarioTermos.class);
28         suite.addTestSuite(Cad_Curso.class);
29         suite.addTestSuite(Cad_Docente.class);
30         suite.addTestSuite(Cad_Enquete.class);
31         suite.addTestSuite(Cad_Duvida.class);
32         suite.addTestSuite(Cad_Equipe.class);
33         suite.addTestSuite(Cad_Feedback.class);
34         suite.addTestSuite(Cad_Filial.class);
35         suite.addTestSuite(Chat.class);
36         suite.addTestSuite(Cad_Modulos.class);
37         suite.addTestSuite(Cad_Outras_Avaliacoes.class);
38         suite.addTestSuite(Cad_Segmento.class);
39         suite.addTestSuite(Cad_Turma.class);
40         suite.addTestSuite(Cad_Pefis_Acesso.class);
41         suite.addTestSuite(Cad_Programa.class);
42
43         //Smoke tests Ambiente Aluno
44         suite.addTestSuite(Ajuda_Suporte.class);
45         suite.addTestSuite(Catalogo_Cursos.class);
46         suite.addTestSuite(Central_de_Mensagens.class);
47         suite.addTestSuite(Comunidades.class);
48         suite.addTestSuite(Expandir_Meu_Espaco.class);
49         suite.addTestSuite(Forum.class);
50         suite.addTestSuite(Home.class);
51         suite.addTestSuite(Meu_Treinamento.class);
52         suite.addTestSuite(Midias_Digitais.class);
53         suite.addTestSuite(Outros_Cursos.class);
54         suite.addTestSuite(Pre_Inscrição.class);
55         suite.addTestSuite(Programas.class);
56         suite.addTestSuite(Termos_de_Uso.class);
57         suite.addTestSuite(WebTV.class);
58
59
60         return suite;
61 }
62
63     public static void main(String[] args) {
64         junit.textui.TestRunner.run(suite());
65     }
66 }
```

Fonte: o autor

Na figura 22 são exibidos os *scripts* utilizados a cada *build* aplicado em homologação para realizar o *smoke test* nas principais funcionalidades do sistema, chamado pela equipe de mini regressão.

Figura 22 – Mini Regressão

Source Control | Executa_MiniRegressao.java

```
283
284    @Test
285    public void testMR_Marina() throws Exception
286    {
287        f_lms.logarAmbienteGestor("ti9", "5fd4g");
288        //// Marina ////
289            f_lms.addSegmento("Segmento Marina");
290
291            f_lms.addAluno("Aluno Marina", "marina.moraes@webaula.com.br", "69905584668", "Segmento Marina");
292            f_lms.addAluno("Aluno Marina_2", "marina.moraes2@qawebaula.com.br", "26138154185", "Segmento Marina");
293
294            f_lms.addCursoOnline("Curso Online Marina", "Informática", "", "Segmento Marina", "Coordenador Geral");
295            f_lms.addCursoPresencial("Curso Presencial Marina", "Informática", "Segmento Marina", "Coordenador Geral");
296
297            f_lms.addTopico("Curso Online Marina", "Tópico 1", "wafile02/QA$/Uteis/imagens/conteudo1.jpeg");
298            f_lms.addTopico("Curso Online Marina", "Tópico 2", "wafile02/QA$/Uteis/imagens/conteudo2.jpg");
299            f_lms.addTopico("Curso Online Marina", "Tópico 3", "wafile02/QA$/Uteis/imagens/conteudo3.jpg");
300
301            f_lms.addDocente("Docente Marina", "marina.moraes@webaula.com.br", "67078256581", "Curso Online Marina");
302
303            f_lms.addTurmaCursoOnl("Turma Online Marina", "Curso Online Marina", "Docente Marina");
304            f_lms.addTurmaCursoPres("Turma Presencial Marina", "Curso Presencial Marina", "Docente Marina");
305
306            f_lms.matric_Turma_Curso("Curso Online Marina", "Turma Online Marina", "Aluno Marina", "marina.moraes@webaula.com.br");
307            f_lms.matric_Turma_Curso("Curso Online Marina", "Turma Online Marina", "Aluno Marina_2", "marina.moraes2@qawebaula.com.br");
308
309            f_lms.matric_Turma_Curso("Curso Presencial Marina", "Turma Presencial Marina", "Aluno Marina", "marina.moraes@webaula.com.br");
310            f_lms.matric_Turma_Curso("Curso Presencial Marina", "Turma Presencial Marina", "Aluno Marina_2", "marina.moraes2@qawebaula.com.br");
311
312            bancoQuestoes("Curso Online Marina");
313            bancoQuestoes("Curso Presencial Marina");
314            cadastraAvaliacaoOnline("Curso Online Marina", "Turma Online Marina");
315    }
316
317
```

Fonte: o autor

Para a realização dos testes de regressão após a aplicação do *build* em produção, são executados todos os *scripts* já criados pela equipe até o momento, conforme mostra a figura 23.

Figura 23 – Regressão

Source Control | Executa_Tudo.java

```
1  //SCRIPT QUE EXECUTA TODOS OS TESTES(CLASSES) JÁ CADASTRADOS
13 package executa_Testes;
14
15 import junit.framework.Test;
29
30
31 public class Executa_Tudo
32 {
33
34         public static Test suite()
35         {
36             TestSuite suite = new TestSuite();
37
38             //Alimenta banco
39             suite.addTestSuite(Alimenta_Banco.class);
40
41             //Smoke tests Ambiente Gestor
42             suite.addTestSuite(Cad_Alunos.class);
43             suite.addTestSuite(Blq_Lib_Aluno.class);
44             suite.addTestSuite(Cad_Avaliacoes.class);
45             suite.addTestSuite(Cad_Curso.class);
46             suite.addTestSuite(Cad_Docente.class);
47             suite.addTestSuite(Cad_Segmento.class);
48             suite.addTestSuite(Cad_Turma.class);
49             suite.addTestSuite(Cad_Equipe.class);
50             suite.addTestSuite(Cad_Filial.class);
51             suite.addTestSuite(Cad_Modulos.class);
52             suite.addTestSuite(Cad_Outras_Avaliacoes.class);
53             suite.addTestSuite(Cad_Programa.class);
54             suite.addTestSuite(Cad_Coordenacao.class);
55             suite.addTestSuite(Chat.class);
56             suite.addTestSuite(Cad_Caracteristicas.class);
57             suite.addTestSuite(Cad_Pefis_Acesso.class);
58             suite.addTestSuite(Cad_Feedback.class);
59             suite.addTestSuite(Cad_Clipping.class);
60             suite.addTestSuite(Cad_Contato.class);
61             suite.addTestSuite(Cad_Dicas.class);
62             suite.addTestSuite(Cad_DicionarioTermos.class);
63             suite.addTestSuite(Cad_Enquete.class);
64             suite.addTestSuite(Cad_Duvida.class);
65             suite.addTestSuite(Cad_Banco.class);
66             suite.addTestSuite(Cad_Cargo.class);
67
68             //Smoke tests Ambiente Aluno
69             suite.addTestSuite(Ajuda_Suporte.class);
70             suite.addTestSuite(Catalogo_Cursos.class);
71             suite.addTestSuite(Central_de_Mensagens.class);
72             suite.addTestSuite(Comunidades.class);
73             suite.addTestSuite(Expandir_Meu_Espaco.class);
74             suite.addTestSuite(Forum.class);
75             suite.addTestSuite(Home.class);
76             suite.addTestSuite(Meu_Treinamento.class);
77             suite.addTestSuite(Midias_Digitais.class);
78             suite.addTestSuite(Outros_Cursos.class);
79             suite.addTestSuite(Pre_Inscrição.class);
80             suite.addTestSuite(Programas.class);
81             suite.addTestSuite(Termos_de_Uso.class);
82             suite.addTestSuite(WebTV.class);
83
84             //Suíte de Testes por Recursos
85             //Comunidades
86             suite.addTestSuite(Comunidades_Aluno.class);
87             suite.addTestSuite(Comunidades_Gestor.class);
88             //Equipe e Filiais
89             suite.addTestSuite(Ativar_Desativar_Alunos.class);
90             suite.addTestSuite(Envio_de_Newsletter.class);
91             suite.addTestSuite(Filiais.class);
92             suite.addTestSuite(Liberacao_de_Alunos.class);
93             suite.addTestSuite(ListarAlunosEquipes.class);
94             //Noticias
95             suite.addTestSuite(Noticias.class);
96             //Sala de aula
97             suite.addTestSuite(Sala_de_Aula.class);
98             //Matricula de aluno
99             suite.addTestSuite(RelatMatricTurmaCurso.class);
100
101
102            return suite;
103        }
104
105        public static void main(String[] args) {
106            junit.textui.TestRunner.run(suite());
107        }
108 }
```

Fonte: o autor