



TESTES AUTOMATIZADOS NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARES

Leonardo Roxo Pessanha Izabel

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Antônio Cláudio Gómez de Sousa

Rio de Janeiro

Dezembro de 2014

TESTES AUTOMATIZADOS NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARES

Leonardo Roxo Pessanha Izabel

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO
DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA
POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO
PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE
ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

Leonardo Roxo Pessanha Izabel

Orientador:

Prof. Antônio Cláudio Gómez de Sousa, Dr.

Examinador:

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Examinador:

Prof. Ricardo Rhomberg Martins, D. Sc.

Rio de Janeiro – RJ, Brasil

Dezembro de 2014

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica – Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro – RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmear ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es).

DEDICATÓRIA

Aos meus pais, irmão, madrinha e amigos que tanto me ajudaram na longa caminhada que resultou neste trabalho e na conclusão do curso.

AGRADECIMENTO

Ao povo brasileiro que contribuiu de forma significativa à minha formação e estada nesta Universidade. Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

RESUMO

O projeto consiste em um estudo sobre a automação de testes durante o desenvolvimento de um software. Para nos aprofundarmos nesse tema, realizamos um estudo sobre todos os principais conceitos e os diferentes tipos de testes.

Além disso, selecionamos algumas das mais utilizadas ferramentas do mercado de modo a entender um pouco mais seu funcionamento e suas aplicações na área de automação de testes.

Com o objetivo de ilustrar o funcionamento de um teste automatizado, exemplificamos, através de um estudo de caso, a construção e o funcionamento de um projeto automatizado.

Palavras-Chave: testes, automação, software, qualidade.

ABSTRACT

The project consists of a study of the automation of testing during the development of software. To understand better this subject, we conducted a study on all the key concepts and the different types of tests.

In addition, we selected some of the most used tools in the market in order to understand its operation and its applications in test automation area.

In order to illustrate the operation of an automated test, we exemplified it by constructing a case study.

Key-words: test, automation, software, quality.

Sumário

Capítulo 1	1
Introdução	1
1.1. Título	1
1.2. Ênfase	1
1.3. Tema	1
1.4. Delimitação	1
1.5. Localização	1
1.6. Justificativa	2
1.7. Objetivo	2
1.8. Metodologia	3
Capítulo 2	4
Testes e Qualidade (Software)	4
2.1. A necessidade de se testar um software	4
2.2. Definição de Teste	5
2.3. Conceitos	7
2.3.1. Erro (“Error”)	7
2.3.2. Defeito (“Defect”)	7
2.3.3. Falha (“Failure”)	7
2.3.4. Verificação	8
2.3.5. Validação	8
2.3.6. Teste	8
2.3.7. Depuração	9
2.4. Modelos / Métodos	9
2.4.1. Modelo Caixa-Preta	9
2.4.2. Modelo Caixa-Branca	10
2.4.3. Modelo Caixa-Cinza	10
2.5. Tipos de Teste	10
2.5.1. Testes Unitários	11
2.5.2. Testes de Integração	11
2.5.3. Testes de Sistema	11

2.5.4.	Testes Funcionais	12
2.5.5.	Aceitação	12
2.5.5.1.	Testes Alfa	13
2.5.5.2.	Testes Beta	13
2.5.6.	Testes de Regressão	13
2.5.7.	Testes Não-Funcionais	13
2.5.7.1.	Testes de Desempenho	13
2.5.7.1.1.	Testes de Carga	14
2.5.7.1.2.	Testes de Stress	14
2.5.7.2.	Testes de Segurança	15
2.5.7.3.	Testes de Portabilidade	15
Capítulo 3		17
Automação de Testes		17
3.1.	Definição	17
3.2.	Quando não automatizar?	17
3.3.	Casos para automatização de testes	18
3.3.1.	Testes de Carga e Desempenho	18
3.3.2.	Testes de Regressão	18
3.3.3.	Testes manuais repetitivos	18
3.3.4.	Testes de Unidade	18
3.3.5.	Preparação de pré-condições	18
3.4.	Ferramentas de Automação	19
3.4.1.	JUnit	19
3.4.2.	Selenium 2.0	19
3.4.3.	Jenkins	21
3.4.4.	Git	22
3.4.5.	GitHub	22
3.4.6.	Eclipse	22
3.4.7.	Ant	22
Capítulo 4		23
Online Bank Management System		23
4.1.	Definição	23
4.2.	Arquitetura	23
4.2.1.	Aplicação Web	23

4.2.2.	Banco de dados.....	24
4.3.	Fluxos de funcionamento do Sistema.....	25
4.3.1.	Autenticação.....	25
4.3.2.	Cadastro de novo usuário	25
4.3.3.	Boas vindas	26
4.3.4.	Criar conta.....	27
4.3.5.	Depositar	28
4.3.6.	Sacar	28
4.3.7.	Verificar saldo.....	29
4.3.8.	Movimentação entre contas.....	30
4.3.9.	Relatório de movimentações	31
4.4.	Casos de testes.....	31
Capítulo 5	33
Automação de Teste - Estudo de Caso.....		33
5.1.	Visão Geral.....	33
5.2.	Implementação da automação de testes de Interface com o Selenium.....	33
5.3.	Implementação da automatização de testes com o Ant.....	35
5.4.	Integração Contínua com Jenkins.....	38
5.5.	Manutenção do ciclo de Integração Contínua (CI)	41
Capítulo 6	42
Resultados e Conclusões		42
6.1.	Resultados	42
6.2.	Conclusão.....	43
Referências		44
Bibliografia		45
Apêndice A.....		46
Casos de Teste.....		46

Lista de Figuras

Figura 1 – Erro x Defeito x Falha	8
Figura 2 – Modelo caixa preta.....	9
Figura 3 – Modelo caixa branca	10
Figura 4 – Selenium IDE.....	20
Figura 5 – Exportação de casos de testes	20
Figura 6 – Modelo de comunicação entre a página web, o servlet e o server	24
Figura 7 – Modelo de comunicação entre a aplicação Java e o banco ODBC.....	24
Figura 8 - Imagem ilustrando tela de autenticação	25
Figura 9 - Imagem ilustrando tela de registro de novo usuário	26
Figura 10 - Imagem ilustrando tela de boas vindas	27
Figura 11 - Imagem ilustrando tela de criação de conta.....	27
Figura 12 - Imagem ilustrando tela de depósito	28
Figura 13 - Imagem ilustrando tela de saque.....	29
Figura 14 - Imagem ilustrando tela de requisição de saldo	29
Figura 15 – Imagem ilustrando tela de exibição de saldo.....	30
Figura 16 - Imagem ilustrando tela de transferências	30
Figura 17 – Imagem ilustrando tela do relatório de movimentações.....	31
Figura 18 – Configuração de velocidade do Selenium IDE.....	34
Figura 19 – Opção para executar os testes com o JUnit	35
Figura 20 – Configurações básicas do Eclipse para utilizar o Ant	36
Figura 21 – Geração do arquivo build.xml	36
Figura 22 – Configuração da execução do arquivo build.xml	37
Figura 23 – Resultados dos testes	38
Figura 24 – Configurações dos caminhos do JDK e do Ant	39
Figura 25 – Configuração do servidor SMTP	39
Figura 26 – Trigger do GitHub	40
Figura 27 – Configuração do Trigger	40
Figura 28 – Configuração do Ant.....	40
Figura 29 - Configuração do Relatório de JUnit	41
Figura 30 – Configuração de e-mail.....	41

Capítulo 1

Introdução

1.1. Título

Testes Automatizados no Processo de Desenvolvimento de Softwares

1.2. Ênfase

Computação

1.3. Tema

O tema do trabalho é o estudo do processo de testes durante o desenvolvimento de softwares de modo a seguir padrões de qualidade. Nesse sentido, pretende-se responder a questão relacionada ao esforço gasto em um processo de teste de um software feito manualmente. A hipótese inicial é que se deve automatizar todo e qualquer teste que venha a ser feito em um determinado software. Dessa forma, sempre que qualquer correção ou atualização seja feita no software será possível avaliar o impacto dessas modificações.

1.4. Delimitação

O objeto de estudo é a automatização de testes de software seguindo padrões de qualidade. A análise das diversas metodologias e ferramentas de testes tem por finalidade orientar em diferentes cenários de testes como proceder com excelência de modo a diminuir esforços.

1.5. Localização

No processo de desenvolvimento de softwares, é comum, antes de colocar um sistema em produção, submeter o software a um processo de avaliação de qualidade. Esse controle de qualidade foi, e em muitos casos ainda é, realizado com auxílio de testes manuais executados por usuários ou mesmo equipes especializadas em testes. Esse tipo de metodologia, quando aplicada a grandes sistemas de software, tem frequentemente se mostrado ineficaz gerando uma excessiva quantidade de erros no

produto, aumentando assim os esforços na sua manutenção. Uma alternativa a essa metodologia é a automatização de testes. Este trabalho localiza-se nesse último método, onde a utilização de ferramentas de automatização de testes permite testes mais eficazes de forma a adequar um software aos padrões de qualidade.

1.6. Justificativa

A cada dia que passa, o mundo moderno está mais imerso no universo computacional. Durante o dia-a-dia, o ser humano se depara constantemente com softwares em quase todos os lugares.

Cada aparelho ou dispositivo portador de um sistema de software teve durante seu processo de desenvolvimento centenas ou até milhares de defeitos encontrados e corrigidos por seus desenvolvedores e testadores. Ainda assim, é comum um usuário encontrar possíveis erros e defeitos nesses sistemas durante a sua utilização. Percebeu-se que com o aumento dos sistemas, a depuração de todos os possíveis erros a cada modificação feita no sistema torna-se uma tarefa de esforço muito elevado. Nesse cenário surgiu o conceito da automatização de testes.

O presente projeto é um estudo dos métodos de testes e padrões de qualidade de serviços na área de desenvolvimento de software. Tal estudo abordará como deve ser feita a automatização de cada tipo de teste analisando a utilização de ferramentas disponíveis no mercado.

Assim, a importância desse trabalho reside na premissa de que em projetos de sistemas de software num ambiente profissional é necessária a utilização de testes automatizados de forma a manter um bom produto (menor número possível de falhas) com preço competitivo.

1.7. Objetivo

O objetivo geral é, então, realizar um estudo que possa reunir todos os métodos de testes contemporâneos e relacioná-los com ferramentas de automação. Dessa forma, tem-se como objetivos específicos: (1) estudar os diferentes tipos de testes existentes; (2) estudar diversas ferramentas de automação existentes no mercado; (3) analisar os melhores casos entre tipo de teste versus ferramenta; (4) realizar um estudo de caso.

1.8. Metodologia

Este trabalho irá abordar diversos tipos de testes em diferentes ferramentas. Tendo em vista esse objetivo, a primeira etapa será realizar um levantamento dos métodos de testes mais produtivos e utilizados em sua respectiva categoria atualmente. Dessa forma, nesse primeiro momento será feita uma pesquisa qualitativa.

Em relação à automatização de testes, será feito um levantamento de algumas ferramentas do mercado que conseguem cobrir os testes estudados previamente. Após esse levantamento, diante das ferramentas pesquisadas, serão escolhidas para a realização de testes aquelas que conseguirem cobrir uma maior quantidade de categorias de testes estudados e possibilitarem sua utilização livre (Open Source) ou possuírem um período de uso para conhecer a ferramenta (Trial).

Por fim, pretende-se aplicar esse conhecimento adquirido realizando um estudo de caso.

O êxito deste trabalho está centrado na determinação de como realizar diversos tipos de testes nas ferramentas mais apropriadas. Tal êxito reside também na realização de um teste completo realizado em um software de código aberto, inserindo possíveis erros e verificando se os resultados ocorrem como esperado.

Capítulo 2

Testes e Qualidade (Software)

Neste capítulo veremos os principais conceitos de testes de software aplicados de modo a garantir padrões aceitáveis de qualidade.

2.1. A necessidade de se testar um software

O objetivo do desenvolvimento de um software é a resolução de um problema real existente no nosso mundo, através de comandos escritos que serão interpretados por um computador.

Ao analisar essa frase já é possível perceber que quando um desenvolvedor propõe-se a construir um software, diversas falhas poderão surgir no andamento desse processo.

Em primeiro lugar, há casos em que a falha ocorre simplesmente na resolução do problema real. O desenvolvedor pode vir a criar uma solução que resolva parte do problema analisado. Consequentemente, estaria ignorando outros cenários que, possivelmente, viriam a ocorrer durante a utilização da ferramenta por outros usuários.

Em segundo lugar, o erro pode ocorrer na escrita dos comandos. As linguagens de programação, como o próprio nome define, são linguagens e, portanto envolvem ambiguidades. Muitos são os casos em que, no nosso dia-a-dia, nós nos deparamos com falhas no entendimento entre humanos por ambiguidade na comunicação. Logo, é comum que isso ocorra também quando tentamos comandar um computador através desse mesmo meio.

Por último, é importante ressaltar que ao se escrever um software para resolver um problema mais elaborado, o grau de complexidade do código também aumenta. Dessa forma, em muitas ocasiões, o próprio desenvolvedor perde o rastro do significado de todas as linhas de seu código. Isso pode vir a ser um problema grave, quando ele efetuar uma modificação em algum pedaço do código e esquecer o impacto da mudança nas outras funções que ele programou. Consequentemente, ele estaria inserindo novos

erros, os quais desconhece, e que só virão a ser detectados posteriormente pelo usuário final.

Em resumo, é perceptível o alto risco de ser produzir um software com defeitos (vulgarmente chamados também por “bugs”). Esse risco é o que justifica a utilização de testes no processo de desenvolvimento. Testar reduz esse risco.

É pensando em reduzir esse risco que se criaram os processos de “Quality Assurance” (QA). Tais processos têm como objetivo produzir softwares de melhor qualidade, ou seja, com o menor índice de erros possível.

É claro que sempre existirão defeitos em um código. Por mais que se teste exaustivamente, nunca é possível verificar todas as possibilidades de entradas e saídas de uma aplicação. Porém, processos de qualidade visam testar um software com o objetivo de se obter um padrão mínimo de qualidade. Assim, o número de defeitos que seria encontrado pelos usuários finais é reduzido, gerando um software melhor.

Em suma, todos os processos de qualidade almejam criar o melhor software possível dentro de um tempo aceitável. Esse software ideal é facilmente entendido citando-se Bruce Sterling em “The Hacker Crackdown”: “The best software is that which has been tested by thousands of users under thousands of different conditions, over years. It is then known as “stable.” This does NOT mean that the software is now flawless, free of bugs. It generally means that there are plenty of bugs in it, but the bugs are well-identified and fairly well understood.”[4]

2.2. Definição de Teste

Testar um software consiste em executá-lo de acordo como foi especificado, para determinar se ele irá comportar como esperado no ambiente para o qual ele foi projetado. O indivíduo que é responsável por essa função é conhecido como *tester* (ou testador se traduzirmos). Cabe ao *tester* a árdua missão de garantir a qualidade do software antes que este vá para o usuário final (ou cliente, quando numa relação comercial).

Para executar essa missão com excelência, um *tester* profissional precisa assumir uma atitude totalmente contrária a de um desenvolvedor. Enquanto o desenvolvedor tenta fazer seu software em construção funcionar perfeitamente, o *tester* possui um

papel destrutivo, tentando, de todas as formas, provar para o desenvolvedor que o programa possui falhas. Por esse motivo, não se recomenda que os testes sejam feitos pelo desenvolvedor.

É claro que um desenvolvedor é capaz de assumir esse papel, porém ele assume uma visão otimista ao criar seu software. Existe uma tendência lógica que faz com que o desenvolvedor acredite que suas modificações estão sempre aperfeiçoando o software, criando a solução perfeita para o problema inicial. Porém, isso é apenas uma suposição, que se não for devidamente testada, não agrega nenhuma qualidade ao produto, resultando assim em usuários insatisfeitos com defeitos no software. Além disso, estudos empíricos comprovam que 50 % das modificações feitas no código pelo desenvolvedor na verdade inserem outros defeitos no código em diversos outros pontos do software.

Ao executar um teste, o profissional de teste parte da premissa que o software não está funcionando corretamente. Ele assume o fato de que o sistema está comprometido com diversos defeitos e que é o seu papel desvendá-las antes que um usuário o faça.

Essas duas mentalidades citadas (testador e desenvolvedor) são totalmente contraditórias e conflituosas, tornando-se muito complicado para apenas uma pessoa assumir os dois papéis no mesmo projeto.

Por outro lado, apesar de ser recomendável que o desenvolvedor não assuma o papel de *tester* em um mesmo projeto, ele deve sim testar seu software desde o início da codificação. Isso significa que, por mais que tenhamos um profissional focado em testar, cabe ao desenvolvedor verificar seu código ao máximo, de modo a minimizar os defeitos existentes que viriam a ser encontradas mais a frente no processo de teste.

Segundo Barry Boehm[5], um defeito encontrado no desenvolvimento do design e requisitos da solução custa 10 vezes menos que um encontrado na codificação em si e 100 vezes menos do que um encontrado na entrega do produto. Portanto, o quanto antes os desenvolvedores e os *testers* começarem a testar o produto, menos custoso o projeto será e terá, definitivamente, muito mais qualidade.

Em suma, para atingir bons padrões de qualidade espera-se que no desenvolvimento de um software existam diversas etapas com diversos tipos de testes que virão a ser executados, não somente pelo *tester*, mas também pelo desenvolvedor.

2.3. Conceitos

Existem alguns conceitos importantes que compõem o universo de teste de software. Por mais simples que eles possam parecer, em muitos casos eles se confundem devido à ambiguidade de sentidos que podem assumir na linguagem do dia-a-dia humano. Porém, dentro do escopo de engenharia de software, cada termo que será citado abaixo tem seu próprio significado.

2.3.1. Erro (“Error”)

Um erro é um desvio do que foi concordado pela especificação de requisitos. O erro de um desenvolvedor pode gerar defeitos no software.

2.3.2. Defeito (“Defect”)

Um defeito é inserido no código de um software quando um desenvolvedor comete algum erro. Dessa forma, apenas um erro pode gerar vários defeitos em um determinado código ou vários erros podem vir a causar apenas um defeito.

2.3.3. Falha (“Failure”)

Uma falha é um comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ter sido causada por diversos defeitos e alguns defeitos podem nunca causar uma falha.



Figura 1 – Erro x Defeito x Falha

Fonte: Artigo Engenharia de Software - Introdução a Teste de Software [1]

2.3.4. Verificação

Verificar um software consiste em aferir que o produto é internamente consistente. Ou seja, significa garantir que o software atende à especificação de requisitos. A grande maioria dos testes consiste em testes de verificação, em que o produto é verificado em circunstâncias especificadas para garantir a saída esperada. Em resumo, quando se decide verificar um software, deve-se fazer a seguinte pergunta: “Estamos construindo corretamente o produto?”.

2.3.5. Validação

Validar um software consiste em garantir que o produto está de acordo com as expectativas do cliente ou usuário. Validar vai além de apenas verificar se o sistema está de acordo com as especificações de requisitos de modo a aferir que o sistema faz realmente aquilo que o usuário final espera que ele faça. Em resumo, quando se decide validar um software, deve-se fazer a seguinte pergunta: “Estamos construindo o produto correto?”.

2.3.6. Teste

Teste é um processo para executar um software ou sistema com o objetivo de revelar a presença de defeitos. Com esse objetivo, o teste tem como meta aumentar a confiança sobre o software testado.

2.3.7. Depuração

Depurar é um processo posterior ao teste. Ele ocorre após o teste encontrar alguma falha no sistema ou no software testado e consiste em uma busca no código fonte ou na especificação do software para encontrar e corrigir a falta ou defeito que gerou esse erro.

2.4. Modelos / Métodos

Existem basicamente três diferentes modelos de teste que podem ser executados de modo a aferir o funcionamento de um software:

2.4.1. Modelo Caixa-Preta

A técnica consiste em testar sem que se possua nenhum conhecimento do funcionamento interior da aplicação que está sendo testada. O testador não possui conhecimento da arquitetura do sistema e não tem acesso ao código fonte. Normalmente, quando um testador está executando um teste com o modelo caixa-preta, ele interage apenas com a interface do sistema, provendo entradas e examinando as saídas sem saber como e onde essas entradas estão sendo utilizadas.

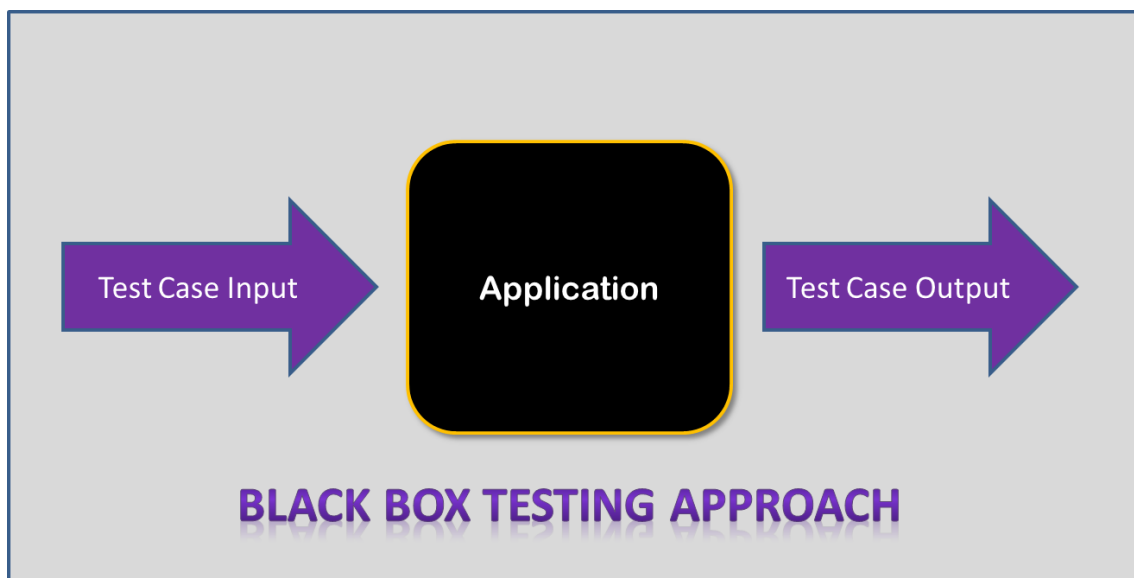


Figura 2 – Modelo caixa preta

Fonte: Understanding White box Testing and Black box Testing Approaches [2]

2.4.2. Modelo Caixa-Branca

A técnica consiste em executar uma investigação da lógica interna e da estrutura do código fonte. O modelo de teste caixa-branca é também conhecido como “glass testing” (teste de vidro) ou teste caixa aberta. Para conseguir executar um teste caixa branca em uma aplicação, o testador precisa possuir um conhecimento do funcionamento interno do código fonte.

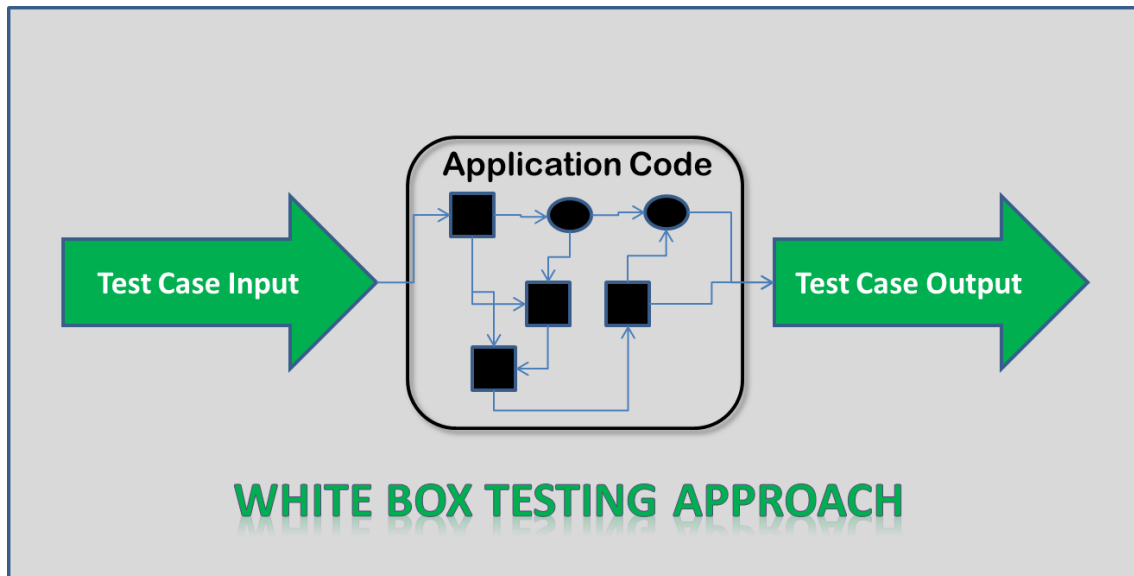


Figura 3 – Modelo caixa branca

Fonte: Understanding White box Testing and Black box Testing Approaches [2]

2.4.3. Modelo Caixa-Cinza

O modelo caixa-cinza é oriundo dos métodos caixa-preta e caixa-branca. O modelo consiste em uma mescla de ambos os casos, adequando-os ao teste de maneira a gerar um teste mais efetivo.

Apesar de ser oriundo da mistura de ambos, o modelo tem uma característica importante: é feito pela interface somente, ou seja, apesar de ele considerar o código durante o planejamento do teste, durante a execução se utilizam apenas as entradas e saídas do modelo caixa preta.

2.5. Tipos de Teste

O planejamento dos testes deve ocorrer em diferentes níveis e em paralelo ao desenvolvimento do software. Abaixo são definidos alguns tipos de teste de software:

2.5.1. Testes Unitários

Testes Unitários têm como objetivo averiguar as menores unidades de software desenvolvidas. Formalmente o IEEE o define como: “Atividade capaz de testar unidades de hardware ou software ou grupo de unidades relacionadas”.

Para realizar esse tipo de teste, deve-se primeiramente selecionar qual será a unidade. Tal escolha é uma decisão de projeto, não existindo, portanto, um caminho único e correto a ser seguido. Por exemplo, um testador de um software que utiliza uma linguagem orientada a objeto pode escolher seus métodos como menor unidade em determinado projeto. Em outros casos talvez não seja necessário tal nível de detalhamento e o testador poderá utilizar as próprias classes como a unidade do teste.

O indivíduo responsável pela execução deste tipo de teste é o próprio desenvolvedor, principalmente devido ao fato de ser um teste caixa-branca. Ou seja, o testador precisa visualizar o código de modo a preparar e executar o teste.

2.5.2. Testes de Integração

Testes de Integração têm como objetivo verificar como cada parte do software se relaciona com as outras. Em outras palavras o IEEE o define como: “Teste no qual os componentes de software, componentes de hardware, ou ambos são combinados e testados para avaliar a interação entre eles”.

Para realizar esse tipo de teste, existem duas estratégias que podem ser seguidas: “Bottom-Up integration” ou “Top-Down integration”. Elas definem o caminho que o testador pretende seguir na integração partindo dos membros de nível mais baixos ou de nível mais alto. Novamente tal decisão é uma escolha de projeto.

Testes de Integração são executados na maioria dos casos pelos próprios desenvolvedores, pois é necessário acesso ao código.

2.5.3. Testes de Sistema

Testes de Sistema têm como objetivo identificar as funções que o software deve realizar (especificação dos requisitos, casos de uso). O IEEE o define como: “Teste conduzido em um sistema completo e integrado para avaliar se o sistema está de acordo com os requisitos especificados”.

Testes de Sistema devem ser executados por uma equipe de teste externa, ou seja, nenhum desenvolvedor deve participar desse teste. É um teste de caixa-preta, ou seja, o testador não precisará conhecer o interior do software para executá-lo.

O teste irá consistir na simples utilização do software em diversas condições de modo a verificar seu comportamento em cada caso.

Os testes de sistema se dividem em duas novas classes de teste: Testes Funcionais e Testes Não-Funcionais.

2.5.4. Testes Funcionais

Testes funcionais têm como objetivo verificar se o sistema está funcionando de acordo como se espera independente da estrutura do sistema, ou seja, nesse caso qualquer fator de interferência externa fora do esperado do sistema não deve ser levado em conta.

Testes funcionais são basicamente o modelo de teste caixa-preta aplicado aos casos de testes que aferem as funcionalidades do sistema ou software que está sendo desenvolvido.

2.5.5. Aceitação

Teste de Aceitação tem como objetivo aferir se o software está de acordo com os padrões de qualidade mínimos definidos na especificação de modo que seja aceito pelo cliente ou usuário.

Em alguns casos, o cliente do projeto utiliza alguns dos usuários finais do produto com vistas a aferir se o produto está satisfatório (UAT -“User Acceptance Testing”).

Testes de aceitação envolvem em muitos casos os próprios artefatos gerados além do código em si. Manuais, documentações, material de treinamento, entre outros artefatos são testados de modo a perceber se o projeto, e não somente o produto, tem qualidade.

Usualmente os testes de aceitação são feitos somente após a consolidação de todo o produto, ou seja, após todos os testes funcionais e não-funcionais terem sido executados e os erros encontrados no processo, corrigidos.

Existem dois subtipos de testes que se enquadram como testes de aceitação:

2.5.5.1. Testes Alfa

Consistem no teste de aceitação feito por um número restrito e escolhido de usuários. É executado dentro do ambiente de desenvolvimento, na maioria dos casos.

2.5.5.2. Testes Beta

Consistem no teste de aceitação feito em maior escala, onde os testadores são os próprios usuários. Os testes beta são normalmente a última fronteira antes da entrega de um software para o cliente ou para o mercado em si.

2.5.6. Testes de Regressão

Testes de Regressão não são exatamente um tipo de teste e sim mais uma estratégia para a redução de efeitos colaterais. Consistem em aplicar, a cada nova versão do software ou a cada ciclo, todos os testes que já foram aplicados nas versões ou ciclos de teste anteriores do sistema. Podem ser aplicados em qualquer nível de teste.

Dessa forma, evita-se que ao modificar ou consertar um erro o desenvolvedor insira um erro crítico que afetará os casos de testes já executados.

Por ser um teste longo (soma de todos os testes já executados) e por ser executado diversas vezes (a cada criação de versão) ele se torna um dos maiores argumentos da automação de testes.

2.5.7. Testes Não-Funcionais

Testes não-funcionais têm como objetivo aferir o funcionamento do sistema em condições extremas. Por se tratarem de condições incomuns o universo de testes não-funcionais é muito grande. Abaixo serão definidos os principais tipos de testes não funcionais:

2.5.7.1. Testes de Desempenho

Testes de desempenho, como o próprio nome já diz, têm como objetivo validar se o sistema funciona em situações mais próximas na realidade a qual ele será exposto. O teste tem como meta aferir se o software reagirá nos tempos esperados mediante

pedidos e acessos de mais de um usuário. Além disso, mede-se como o software reagiria em condições mais extremas de falta de conexão, ou intermitência na rede.

Os testes de desempenho se dividem em dois tipos de testes:

2.5.7.1.1. Testes de Carga

Testes de carga consistem na simples verificação do comportamento do sistema mediante a aplicação de uma carga máxima de acessos e manipulações de dados de entrada.

Podem ser executados em condições normais, simulando a realidade em que se pretende inserir o sistema, ou em condições de pico, simulando condições extremas que podem vir a ocorrer devido a alguma anomalia.

Testes de carga são executados, na grande maioria dos casos, com a ajuda de ferramentas automáticas como Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test etc.

Nessas ferramentas são definidos usuários virtuais que executaram scripts selecionados de maneira a efetuar o teste de carga no software. A quantidade de usuários é facilmente ajustada em suas configurações de acordo com os requisitos do sistema.

2.5.7.1.2. Testes de Stress

Testes de stress visam aferir o comportamento do software sob condições anormais. Durante a execução de um teste de stress retiram-se recursos aplicando uma carga acima dos limites definidos pelos requisitos.

O principal foco é testar a aplicação em condições extremas de modo a encontrar o ponto no qual o software deixa de funcionar, conhecida como “breaking point”.

Testes de stress podem ser executados em diferentes cenários, abaixo se encontram alguns deles:

- Conexão intermitente com o banco de dados
- Conexão intermitente na rede

- Aplicando carga através de diferentes processos que consomem memória, processamento no servidor.
- Aplicando carga através de diferentes processos que consomem memória, processamento no cliente.

2.5.7.2. Testes de Segurança

Testes de segurança são um tipo de teste que tem como objetivo expor todas as possíveis vulnerabilidades que um sistema possui. Dessa forma, pode-se determinar se os dados e recursos do seu software estão devidamente protegidos contra um possível ataque externo.

Existem 4 diferentes áreas que devem ser consideradas durante a execução de um teste de segurança:

- Segurança da Rede: visa descobrir vulnerabilidades da infraestrutura da rede do sistema.
- Segurança dos softwares do sistema: visa descobrir vulnerabilidades nos softwares, com os quais o sistema em questão se comunica e dos quais depende.
- Segurança da aplicação no cliente: visa garantir que o cliente não seja manipulado.
- Segurança da aplicação no servidor: visa garantir que o servidor é robusto o suficiente de modo a não permitir qualquer acesso de intrusos.

Existem infinitas formas de invadir um sistema, logo existem infinitos tipos e formas de proteger e testar um software, de forma a garantir sua segurança. Por mais difícil que seja criar um software totalmente seguro contra tudo, dependendo da criticidade da funcionalidade do software, deve-se utilizar ao máximo esse tipo de teste.

2.5.7.3. Testes de Portabilidade

Testes de portabilidade, como do próprio nome se infere, visam validar se o software desenvolvido pode ser utilizado em diferentes ambientes. Hardwares, sistemas operacionais e navegadores costumam ser os maiores focos dos testes de portabilidade.

Tais testes têm grande valor para sistemas multi-plataforma que tentam atingir todos os tipos de clientes com sua aplicação.

Dessa forma, vemos abaixo alguns exemplos básicos mais utilizados de testes de portabilidade:

- Transferir o software de um computador para outro;
- Construir o executável para testar o software em diferentes plataformas de sistema operacional;
- Utilizar diferentes navegadores para testar a aplicação.

Testes de portabilidade exigem que o sistema em si tenha sido construído com o intuito de ser portátil.

Capítulo 3

Automação de Testes

Neste capítulo, estudaremos os principais casos e aplicações para automação de testes, assim como algumas ferramentas largamente utilizadas nesses processos.

3.1. Definição

Automação de testes, por definição, consiste em utilizar um software externo (que não faça parte do software que está sendo testado) para controlar a execução dos testes e a comparação dos resultados encontrados com os resultados esperados. Através desse processo, podem-se automatizar algumas tarefas de teste repetitivas, porém necessárias, que fazem parte do fluxo de teste de um software ou adicionar novos testes que seriam muito trabalhosos, e consequentemente custosos para serem executados manualmente.

3.2. Quando não automatizar?

A automação de testes tem como objetivo reduzir esforços em tarefas repetitivas, logo nem todos os testes devem ser automatizados. Antes de se inserir esse processo em um projeto deve-se analisar qual é o volume de testes que seu software necessita e o tempo que é gasto para executar essas baterias de testes. Se o projeto em questão tiver uma complexidade muito pequena de modo que o volume de teste para aferir a qualidade do sistema é muito baixo, o esforço para inserir essa metodologia de testes pode ser mais custoso do que a manutenção dos testes manuais. Somado a isso, o tempo para a execução dos testes automáticos não seriam, nesse caso, tão menores que os manuais.

Outro fator importante é que o teste automatizado, como qualquer outro software, comunica-se com o software que está sendo testado através de uma interface. Se essa interface sofre constantes modificações, seus testes sofrerão alterações também. Esse é o pior cenário para a automação de testes. Em casos como esse a automação deve ser evitada, pois os testes automáticos serão constantemente alterados manualmente, contrariando totalmente os princípios do processo em si.

3.3. Casos para automatização de testes

3.3.1. Testes de Carga e Desempenho

Testes automáticos são um pré-requisito para esses tipos de teste. É impraticável economicamente utilizar 100 ou mais usuários, durante um teste, para acessar um sistema simultaneamente a fim de avaliar a qualidade do sistema sob tais condições.

3.3.2. Testes de Regressão

É uma das melhores aplicações da automação de testes. Testes de regressão usualmente custam muito tempo para serem executados manualmente e são muito suscetíveis a erro humano devido, ao seu nível de repetitividade. Ao automatizar, reduz-se drasticamente a possibilidade de o sistema ir para produção com a inserção de um novo defeito em uma funcionalidade antiga. Além disso, o ganho em tempo de execução dos testes geralmente justifica a sua utilização.

3.3.3. Testes manuais repetitivos

Nesses casos, a grande vantagem do teste automático se deve à redução de erros propriamente. Testes manuais repetitivos normalmente exigem um grande nível de concentração do *tester* de modo que ele não se distraia devido a um cansaço psicológico da repetição. Assim sendo, justifica-se a automação nos casos em que se verifica que seus testes têm obtido resultados não verdadeiros devido à falha humana.

3.3.4. Testes de Unidade

Com a existência de tantas ferramentas gratuitas de testes unitários pra quase todas as linguagens, não utiliza-las é simplesmente contra produtivo. O ganho em tempo ao utilizar essas ferramentas justifica sua utilização na maioria dos casos.

3.3.5. Preparação de pré-condições

Ferramentas de automação de testes podem e devem ser usadas para preparar ambientes de teste como condições iniciais ou entradas que virão a ser utilizadas nos testes. É claro que esse jamais será o primeiro passo a ser dado quando se pretende automatizar um software, porém uma vez que se possua a ferramenta deve-se utilizá-la de modo a maximizar a produtividade.

3.4. Ferramentas de Automação

Neste capítulo, veremos, dentre as ferramentas estudadas, as escolhidas para a realização do estudo de caso.

3.4.1. JUnit

O JUnit é um framework open-source, criado por Erich Gamma e Kent Beck, com suporte à criação de testes unitários automatizados na linguagem de programação Java.

O framework facilita na geração do código a realização dos testes automatizados com apresentação de resultados.

Ao utilizar a ferramenta, é possível criar rapidamente e automaticamente um teste para cada método de uma classe, gerando assim um script de teste em questão de segundos. Cada verificação de método é aferida de modo a avaliar se mediante entradas definidas e resultados esperados a classe funciona da forma esperada.

Uma vez feito isso, os testes são executados rapidamente sem que, para isso, seja interrompido o processo de desenvolvimento. Ao final, o JUnit checa os resultados dos testes e fornece uma resposta imediata exibindo possíveis falhas.

Além disso, permite criar uma hierarquia de testes que possibilitará testar apenas uma parte do sistema ou todo ele.

Somado a isso, o JUnit é amplamente utilizado pelos desenvolvedores da comunidade código-aberto, possuindo um grande número de exemplos online que podem ajudar qualquer iniciante nessa área de automação.

3.4.2. Selenium 2.0

Selenium 2.0, ou Selenium WebDriver como também é conhecido, é um framework multi-plataforma para testes de interface em aplicações web desenvolvidos por Jason Huggins em 2004.

O software possui sua própria IDE (Selenium IDE) para gravação e execução de scripts de teste, que o usuário pode utilizar sem qualquer conhecimento de linguagens de programação. O Selenium IDE consiste em um plugin do navegador Firefox.

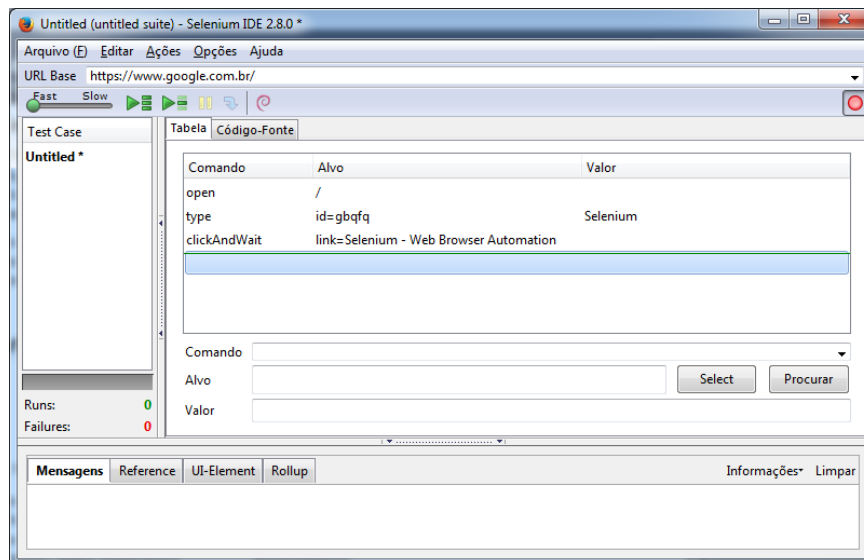


Figura 4 – Selenium IDE

A gravação de testes funciona de maneira simples: basta executar o caso de teste manualmente enquanto o IDE armazena todos os passos seguidos.

Feito isso, o *tester* pode utilizar o próprio IDE para executar o plano de teste utilizando o Firefox como navegador de teste ou exportar os testes em alguma das linguagens possíveis e executá-lo no seu próprio IDE com o navegador que desejar.

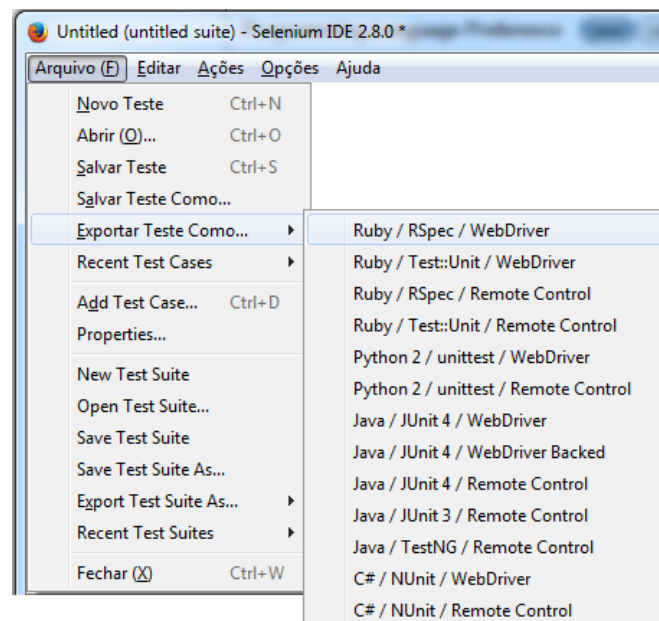


Figura 5 – Exportação de casos de testes

O IDE permite o gerenciamento de vários casos de testes, além de viabilizar a criação de um plano de teste que possua vários casos de teste inclusos.

3.4.3. Jenkins

Jenkins é uma aplicação de código aberto escrita em Java que implementa o conceito de Integração Contínua.

O conceito de Integração Contínua pode ser facilmente entendido citando Martin Fowler: “Integração Contínua é uma pratica de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por um *build* automatizado (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos times acham que essa abordagem leva a uma significativa redução nos problemas de integração e permite que um time desenvolva software coeso mais rapidamente.” [6].

Seguindo esse conceito, o Jenkins atua monitorando a execução de tarefas do dia-a-dia de desenvolvimento alertando aos responsáveis caso algo inesperado ocorra.

Dessa forma, o Jenkins se foca basicamente em:

- Construir *builds* de softwares continuamente – onde se encarrega de gerar um *build* para seu software no intervalo de tempo desejado ou após algum evento desejado.
- Rodar testes
- Executar tarefas externas – possuindo completa integração com diversos sistemas operacionais, é possível programar o Jenkins para executar qualquer tarefa externa sendo ela local ou remota
- Alertar usuários em casos de falhas – ao executar qualquer das outras tarefas citadas acima o Jenkins pode ser utilizado para disparar um e-mail para os responsáveis pelos sistemas alertando possíveis falhas.

Para executar todas essas tarefas o Jenkins dispõe de uma base de milhares de plugins, os quais são utilizados para comunicar o servidor da aplicação com os outros serviços necessários.

3.4.4. Git

Git é um software livre de controle de versão distribuído desenvolvido por Linus Torvalds para o desenvolvimento do Linux Kernel (núcleo do Linux). Consiste em um sistema de gerenciamento de código fonte que visa acelerar o desenvolvimento.

O Git funciona através da criação de um ou mais repositórios que possuem histórico completo e habilidade total de acompanhamento das revisões, não dependente de acesso a uma rede ou a um servidor central.

3.4.5. GitHub

GitHub é um serviço online de armazenamento compartilhado para projetos de software que utiliza o sistema de controle de versionamento Git. É escrito em Ruby on Rails pelos desenvolvedores da empresa Chris Wanstrath, PJ Hyett e Tom Preston.

3.4.6. Eclipse

Eclipse é um IDE para desenvolvimento na linguagem Java. O projeto Eclipse foi iniciado na IBM que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade. Apesar de ser feito para o desenvolvimento em Java, o IDE suporta várias outras linguagens a partir de plugins.

3.4.7. Ant

Apache Ant é uma biblioteca Java e uma ferramenta de linha de comando com objetivo de conduzir processos descritos em arquivos de *build*. O principal uso do Ant consiste em construir *builds* de aplicações Java.

Ant fornece uma série de tarefas que podem ser feitas durante a construção do *build* de uma aplicação tais como: compilar, testar e executar outras aplicações Java.

A biblioteca do Apache Ant já vem automaticamente na instalação de algumas IDE como por exemplo o Eclipse Luna, que será utilizado durante o estudo de caso deste projeto.

Capítulo 4

Online Bank Management System

Neste capítulo, veremos um pouco do software que servirá de caso de estudo para a automatização de testes deste projeto.

4.1. Definição

“Online Bank Management System” consiste em um sistema bancário simplificado desenvolvido por Tousif Khan, disponível para download no site TechZoo[1]. O projeto tem fins educacionais visando ajudar iniciantes que desejam aprender a utilizar Java Server Pages (ou JSP como é usualmente conhecido).

O software consiste em uma página, na qual qualquer usuário pode se cadastrar para criar sua própria conta bancária. Após uma simples validação de usuário e senha, o usuário terá ao seu dispor algumas das mais corriqueiras ações bancárias como saques, depósitos, transferências, entre outros.

Este protótipo de banco online será utilizado como exemplo para a automação de testes em software a ser estudada posteriormente neste projeto.

4.2. Arquitetura

4.2.1. Aplicação Web

A aplicação web foi inteiramente desenvolvida em Java Server Pages (JSP).

JSP consiste em uma tecnologia que permite ao desenvolvedor de páginas para Internet produzir aplicações, que acessem a um banco de dados, geradas dinamicamente, baseadas em HTML (neste caso) e utilizando a linguagem de programação Java.

Para implantar e executar o Java Server Pages, um servidor web compatível com um container Servlet é necessário.

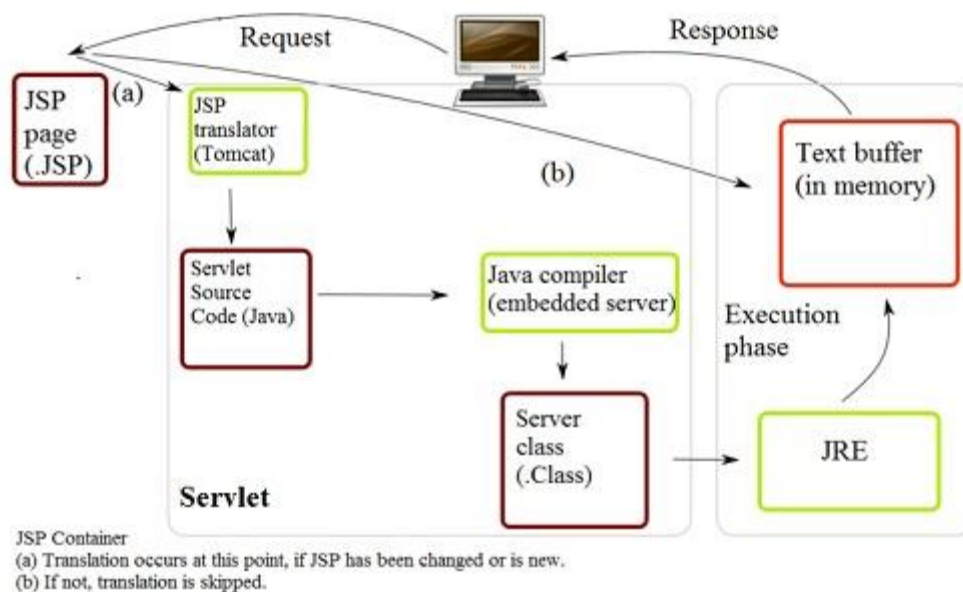


Figura 6 – Modelo de comunicação entre a página web, o servlet e o server

Fonte: Relationship between Servlet and JSP [3]

Essa comunicação (descrita na figura 6) ocorre de maneira que sempre que um cliente fizer um pedido, fazendo o cadastro no site, por exemplo, ele será enviado para o servidor web como um pedido HTML. O servidor web (através do container servlet) irá traduzir esse pedido HTML para JSP, executá-lo e devolver uma resposta em HTML para o cliente.

Nesse caso utilizamos o Apache Tomcat, que o IDE Netbeans 8.0 possui, como servidor web.

4.2.2. Banco de dados

A aplicação utilizou um banco de dados Microsoft Access para armazenar todas as informações relevantes de seus processos.

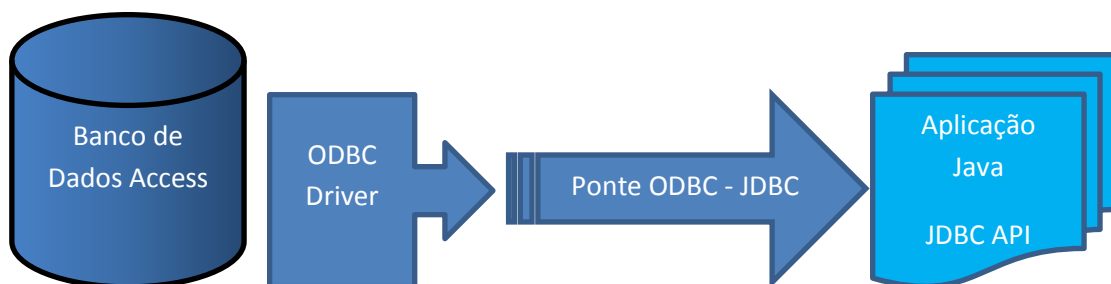


Figura 7 – Modelo de comunicação entre a aplicação Java e o banco ODBC

Como se pode verificar na figura 7, a aplicação faz pedidos através da biblioteca JDBC (Java Database Connectivity), que por meio de uma ponte JDBC – ODBC alcança o driver da ODBC (Open Database Connectivity) da Microsoft, que acessa o banco de dados Microsoft Access encontrado no interior de um arquivo de extensão “mdb”.

4.3. Fluxos de funcionamento do Sistema

O Sistema é basicamente composto por 9 telas diferentes. Abaixo será explicado o funcionamento esperado de cada uma delas.

4.3.1. Autenticação

Ao acessar o endereço do site do banco, o usuário é direcionado para a tela de autenticação. O cliente que já possui cadastro no banco deve inserir seu usuário, sua senha e clicar no botão “Submit”. Dessa forma ele será direcionado para a tela de “Boas Vindas”.

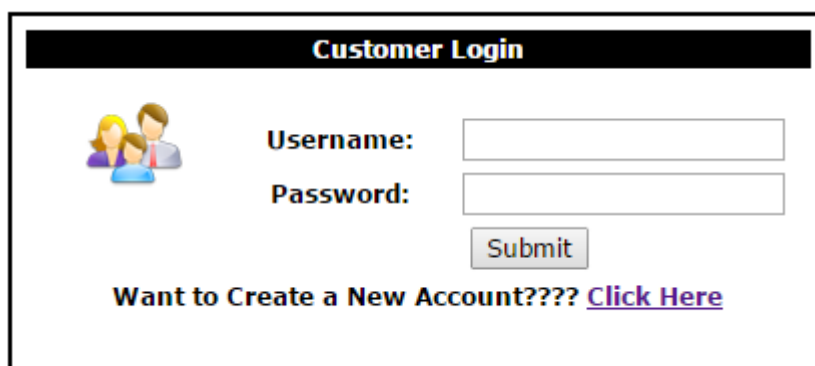
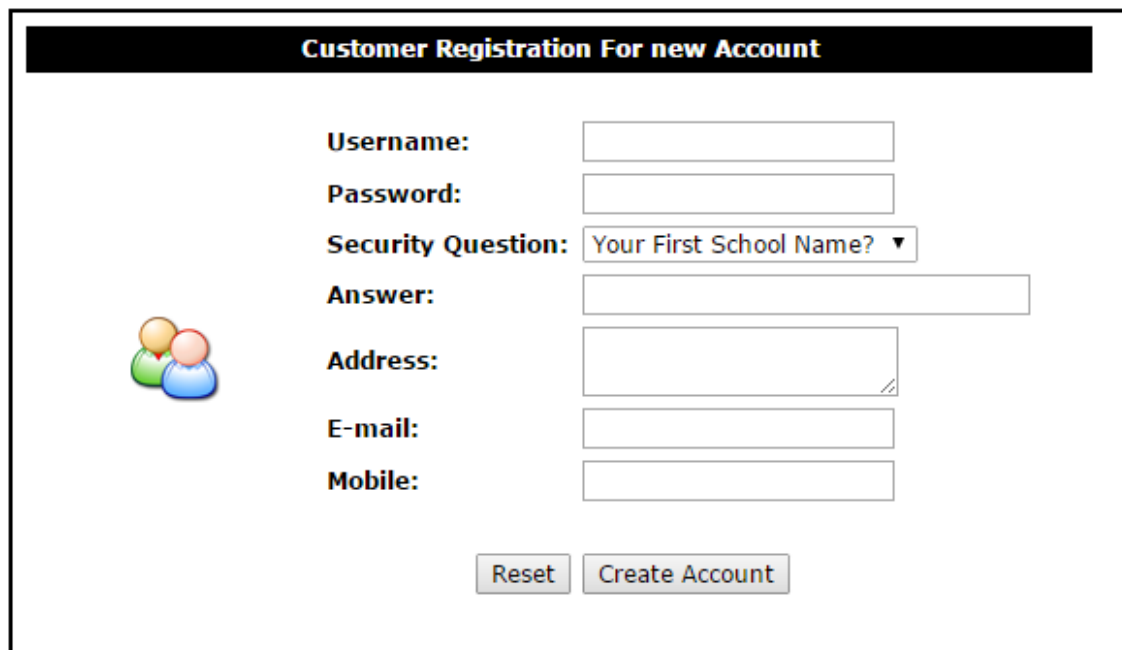
A screenshot of a web form titled "Customer Login". The form has a black header bar with the title in white. Below the header, on the left, is an icon of two people. To the right of the icon are two input fields: "Username:" and "Password:". Below the "Password:" field is a "Submit" button. At the bottom of the form, there is a line of text: "Want to Create a New Account???? [Click Here](#)".

Figura 8 - Imagem ilustrando tela de autenticação

Caso o cliente esteja acessando o endereço do site pela primeira vez, ele deve clicar em “Click Here” de modo a navegar para a tela de “Cadastro de novo usuário”.

4.3.2. Cadastro de novo usuário

Ao clicar no botão “Click Here” da tela de “Autenticação”, o novo cliente será redirecionado para a página de cadastro exibida abaixo:



The image shows a web form titled "Customer Registration For new Account". On the left side of the form is a small icon consisting of three overlapping circles in green, orange, and blue. The form contains the following fields and controls:

- Username:** A text input field.
- Password:** A text input field.
- Security Question:** A dropdown menu with the text "Your First School Name?" and a downward arrow.
- Answer:** A text input field.
- Address:** A text input field with a small icon in the bottom right corner.
- E-mail:** A text input field.
- Mobile:** A text input field.
- Buttons:** Two buttons at the bottom: "Reset" and "Create Account".

Figura 9 - Imagem ilustrando tela de registro de novo usuário

Nessa tela, o usuário deve preencher os campos com seus dados pessoais e, ao final, clicar em “Create Account”. Caso ele deseje apagar os dados inseridos, o cliente deve clicar no botão “Reset”.

Após a criação da nova conta, será exibida uma mensagem de confirmação informando que a conta foi criada com sucesso, ou seja, os dados foram inseridos no banco. Em caso negativo, se houver algum problema de conexão com o banco de dados, por exemplo, o cliente receberá uma mensagem informando o insucesso.

4.3.3. Boas vindas

Nessa tela visualiza-se uma breve explicação dos serviços que o software “Online Bank” pretende oferecer aos seus clientes.

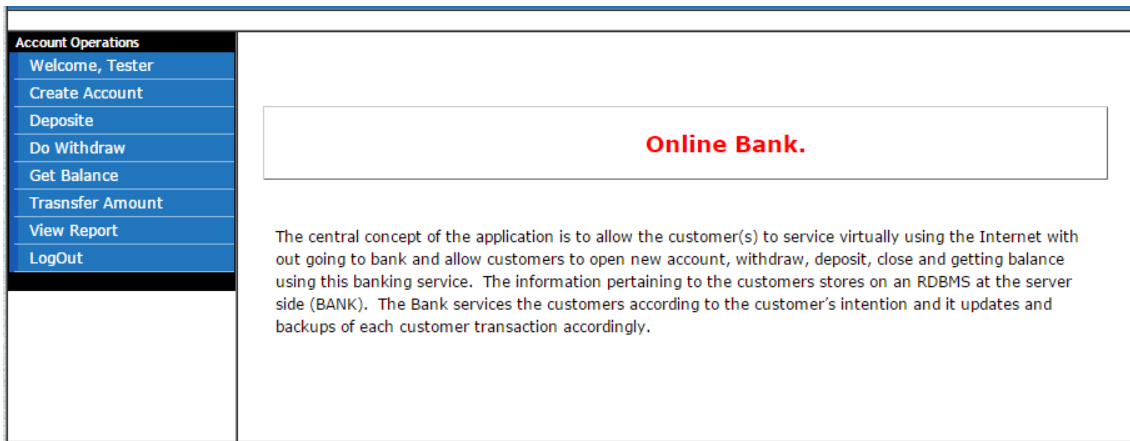


Figura 10 - Imagem ilustrando tela de boas vindas

No lado esquerdo nota-se um menu com todas as opções de ações que o cliente pode executar no banco.

4.3.4. Criar conta

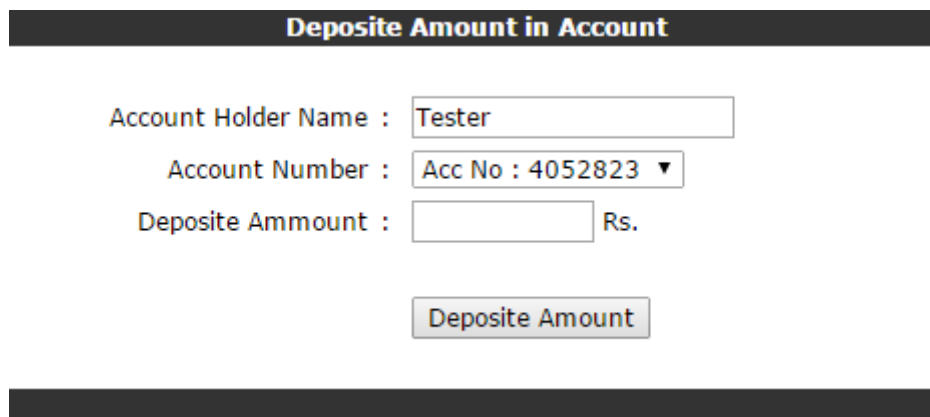
Ao clicar na opção “Create Account” o cliente é indicado para a tela de criação de conta bancária. Nessa tela, o usuário pode escolher o tipo de conta que ele pretende abrir no banco e detalhar qualquer outra informação relevante dessa conta que ele deseje armazenar.

Figura 11 - Imagem ilustrando tela de criação de conta

Ao pressionar o botão “Create Account”, o usuário deverá receber uma mensagem de confirmação da criação da nova conta ou de falha (em caso de algum problema técnico).

4.3.5. Depositar

Ao clicar na opção “Deposit”, o cliente é indicado para a tela de depósito em conta bancária. Nessa tela, o usuário pode escolher em qual conta ele pretende depositar o dinheiro e qual o valor a ser depositado.



Deposit Amount in Account

Account Holder Name :

Account Number :

Deposit Ammount : Rs.

Figura 12 - Imagem ilustrando tela de depósito

Ao pressionar o botão “Deposit Amount”, o usuário deverá receber uma mensagem de confirmação do depósito ou de falha (em caso de algum problema técnico).

4.3.6. Sacar

Ao clicar na opção “Do Withdraw”, o cliente é indicado para a tela de saque de conta bancária. Nessa tela, o usuário poderá escolher de qual conta ele pretende sacar o dinheiro e qual o valor a ser sacado.

Withdraw Amount From Account

Account Holder Name :

Account Number :

Withdraw Ammount : Rs.

Figura 13 - Imagem ilustrando tela de saque

Ao pressionar o botão “Withdraw Amount”, o usuário deverá receber uma mensagem de confirmação de saque ou de falha (em caso de algum problema técnico).

4.3.7. Verificar saldo

Ao clicar na opção “Get Balance”, o cliente é indicado para a tela de verificação de saldo da conta bancária. Nessa tela, o usuário poderá escolher de qual conta ele pretende visualizar o saldo.

Please Select the Account to Check Your Current Balance

Account Holder Name :

Select Account No. :

Figura 14 - Imagem ilustrando tela de requisição de saldo

Ao pressionar o botão “Check Balance”, o sistema irá exibir o saldo na seguinte tela:

Your Current Banalce is As Follows

Customer Name : Tester

Account No. : 4052823

Current Balance : 105 Rs.

If you want to Deposite then [Click Here](#)

Or For Withdraw the Ammount fro Your Account [Click here](#)

Figura 15 – Imagem ilustrando tela de exibição de saldo

4.3.8. Movimentação entre contas

Ao clicar na opção “Transfer Amount”, o cliente é indicado para a tela de transferências bancárias. Nessa tela, o usuário poderá escolher qual a conta de origem da movimentação, a conta destino, qual o valor a ser movimentado e detalhar qualquer outra informação relevante dessa movimentação que ele deseje armazenar.

Transfer Amount

Account Holder Name : Tester

Select Source Account No. : 4052823 ::CURRENT ▼

Select Destination Account No. : 9985493 ::STUDENT ▼

Amount : Rs.

Details :

Transfer Amount

Figura 16 - Imagem ilustrando tela de transferências

Ao pressionar o botão “Transfer Amount”, o usuário deverá receber uma mensagem de confirmação da movimentação ou de falha (em caso de algum problema técnico).

4.3.9. Relatório de movimentações

Ao clicar na opção “View Report”, o cliente é indicado para a tela de “Relatório de movimentações”. Nessa tela, o usuário poderá visualizar todo o histórico de movimentações executada no seu cadastro.

Following are the Repost of Your Acount(s).				
Acc No.	Operation	Amt	Balance	Date - Time
4052823	DEPOSITE	100	100	Fri Nov 28 21:48:48 BRST 2014
4052823	WITHDRAW	5	95	Fri Nov 28 21:48:53 BRST 2014
4052823	WITHDRAW	5	90	Fri Nov 28 21:48:58 BRST 2014
4052823	WITHDRAW	5	85	Fri Nov 28 21:49:01 BRST 2014
4052823	DEPOSITE	20	105	Fri Nov 28 21:49:05 BRST 2014

Figura 17 – Imagem ilustrando tela do relatório de movimentações

4.4. Casos de testes

Por se tratar de um software educacional, sem relacionamentos cliente-desenvolvedor, o desenvolvedor não se preocupou com a geração de nenhum artefato ou documento que explicasse corretamente os requisitos ou formas de uso do nosso sistema. Assim sendo, todo o fluxo do sistema descrito acima foi deduzido empiricamente.

Sem ter em mãos nenhum documento de requisito ou de casos de uso, o que seria altamente inviável em uma situação real, coube a análise de testes desse sistema basear os testes do software no fluxo do sistema já explicado.

Em um primeiro ponto, considerando que boa parte do sistema já está construída focamos nossos casos testes na parte funcional do sistema, seguindo o modelo caixa preta. Dessa forma, qualquer melhoria que venha a ser feita no sistema por um desenvolvedor, protegeria o funcionamento do resto do sistema de qualquer inserção de defeitos.

Em segundo lugar, voltamos nossa atenção para o código, de modo a criar alguns testes caixa branca, para garantir o funcionamento dos principais métodos do sistema. Porém essa tentativa se mostrou improdutiva e falha.

Considerando que o projeto do “Online Bank” já estava implementado e funcionando sem qualquer teste unitário, a inserção desse tipo de teste nos obrigaria a alterar grande parte da estrutura de código. Após algumas análises e estudo no código, concluímos não ser uma boa prática a criação de testes unitários para sistemas de código legado que não tenham tido qualquer preocupação com esse tipo de teste durante a sua construção.

Por se tratar de uma ferramenta web, que seria exposta a qualquer pessoa do mundo via internet, nos preocupamos em realizar casos de testes não-funcionais. Em primeiro lugar, a portabilidade torna-se quase uma exigência para qualquer sistema que será acessado via navegadores. Dessa forma, geramos testes de portabilidade de modo a garantir o funcionamento do sistema em diversos navegadores utilizados ao redor do mundo.

Além disso, por se tratar de um sistema bancário, a segurança deve ser vista como uma das principais virtudes do sistema. Portanto, foram preparados casos de testes que garantissem que o sistema seria seguro contra qualquer intruso.

Por fim, geramos casos de testes de modo a garantir o desempenho do nosso sistema mediante a quantidade de usuários que o sistema viria a ter. Nesse ponto, por esse projeto não visar testes reais, os números de acessos considerados foram pequenos e irreais usados apenas como caso de estudo.

Ainda discutindo desempenho, descartamos casos de testes de stress. Por se tratar de um caso de estudo com poucos recursos de memória e desempenho, qualquer valor obtido nesse tipo de teste não possuiria significado algum.

Todos os testes citados nesse capítulo se encontram no Apêndice A.

Capítulo 5

Automação de Teste - Estudo de Caso

Neste capítulo serão exemplificados, através de um estudo de caso, todos os principais conceitos estudados para a realização e automatização de um projeto de software de uma página web.

5.1. Visão Geral

O estudo que implementaremos consiste em uma aplicação de integração contínua na qual desenvolvedores e testadores poderão trabalhar em conjunto, enquanto testes automatizados serão executados sempre que uma nova versão de código for validada no repositório.

A automação consiste em dois projetos separados: o primeiro diz respeito à aplicação em si (“Online Bank”) e o segundo trata-se de um projeto de testes.

O projeto de testes será configurado com o construtor de arquivos Ant de modo que sempre que uma nova “build” sua for construída, ele execute seus próprios testes em seguida.

Por fim adicionaremos ambos os projetos no Jenkins de modo que a construção do projeto de testes seja atrelada à construção do projeto principal, gerando assim um teste automatizado para todas as novas versões de código para o programa principal. Nos casos de falhas nos testes acionaremos os desenvolvedores através de um e-mail, reportando que erros ocorreram na nova versão do “Online Bank”.

5.2. Implementação da automação de testes de Interface com o Selenium

A utilização do Selenium na automação de testes deste projeto teve como objetivo testar todas as principais funcionalidades da interface do software com o usuário final.

Dessa maneira, o principal foco desta etapa de testes foram os testes de sistema (funcionais). Consequentemente, ao final da preparação de todo o projeto de teste, este artefato se tornará um poderoso teste de regressão.

Como foi dito no capítulo 3, onde foi explicado o funcionamento da ferramenta Selenium WebDriver, utilizamos o Selenium IDE para gravar todos os casos de testes no navegador Firefox.

Após a gravação de todos os casos de testes, executamos todo o caderno de testes (Test Suit) utilizando o próprio Selenium IDE de modo a aferir o funcionamento de todos os testes, bem como se os resultados dos testes condiziam com a realidade do software. Por já possuímos conhecimento da velocidade da execução dos testes nos projetos exportados pelo Selenium no formato de teste unitário do JUnit, utilizamos a velocidade máxima configurável no Selenium IDE na execução.

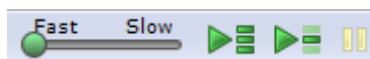


Figura 18 – Configuração de velocidade do Selenium IDE

Dessa forma, podemos garantir que a velocidade de execução do teste nesse momento será a mesma que a do JUnit, quando este executar os testes automáticos posteriormente.

Tendo feito isso, o próximo passo que tomamos foi a exportação dos casos de teste do Selenium. A utilização do conceito de “Test Suit” é importante para a organização dos testes dentro do próprio Selenium IDE, porém não é necessária ao se exportar no formato do JUnit 4. O construtor Ant se encarregará de reunir todos os casos de teste e executá-los.

Dessa maneira, exportamos somente os casos de teste gerando para cada um deles um arquivo com nome do caso de teste e extensão “.java”.

Em seguida, tendo em posse todos os arquivos de teste, iniciamos a configuração do IDE Eclipse, onde a solução responsável pela execução dos testes da aplicação principal (“Online Bank”) será implementada.

Criamos, portanto, um novo projeto no Eclipse IDE, onde adicionamos todos os arquivos de testes ao projeto.

5.3. Implementação da automatização de testes com o Ant

Nesse ponto, já possuímos todos os testes de interface previamente adicionados em uma solução do Eclipse. A seguir, devemos preparar o Eclipse e o projeto de testes de modo a gerar um arquivo de construção (build.xml) que irá executar todos os testes durante esse processo.

Primeiramente, iremos adicionar ao projeto todas as bibliotecas do Selenium WebDriver (todos os arquivos “.jar”), que podem ser obtidos gratuitamente no site oficial do software. Teremos, também, que adicionar a biblioteca do JUnit 4 ao projeto de teste.

Tendo feito isso já será possível executar separadamente cada caso de teste manualmente. Para isso, basta clicar com o botão direito do mouse no arquivo do caso de teste (.java) e selecionar o caminho “Run As -> JUnit Test”, como pode ser visto na figura abaixo:

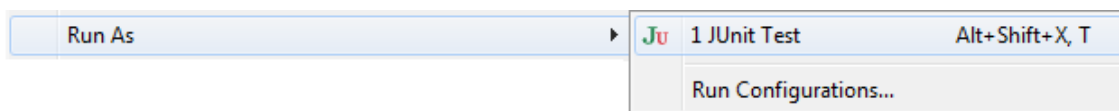


Figura 19 – Opção para executar os testes com o JUnit

O próximo passo que tomamos foi aferir a configuração do Ant Apache dentro do Eclipse. Para isso, basta clicarmos na barra de ferramentas do Eclipse em “Windows -> Preferences” e na janela de preferências que se abrirá quando selecionamos “Ant -> Runtime” no menu do lado esquerdo.

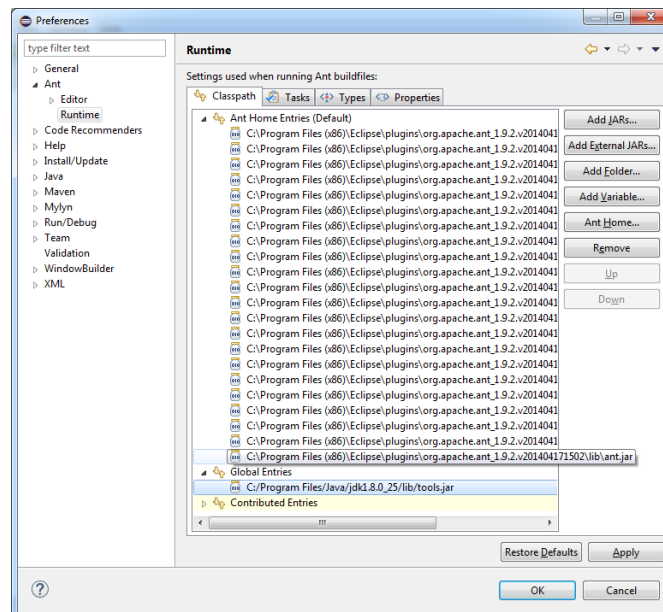


Figura 20 – Configurações básicas do Eclipse para utilizar o Ant

A aba “Classpath” deve possuir dentro de “Ant Home Entries” todos os arquivos “.jar” de biblioteca do Ant Apache. Dentro de Global Entries deve constar o arquivo “tools.jar” do Java Development Kit (JDK). Caso o Eclipse já não tenha essas configurações feitas por padrão, devemos prepará-las manualmente antes de prosseguir.

Em seguida, criaremos o arquivo “build.xml” que utilizará o Ant Apache descrito anteriormente. Para isso, basta clicar com o botão direito no nome do projeto e selecionar a opção “Export”.

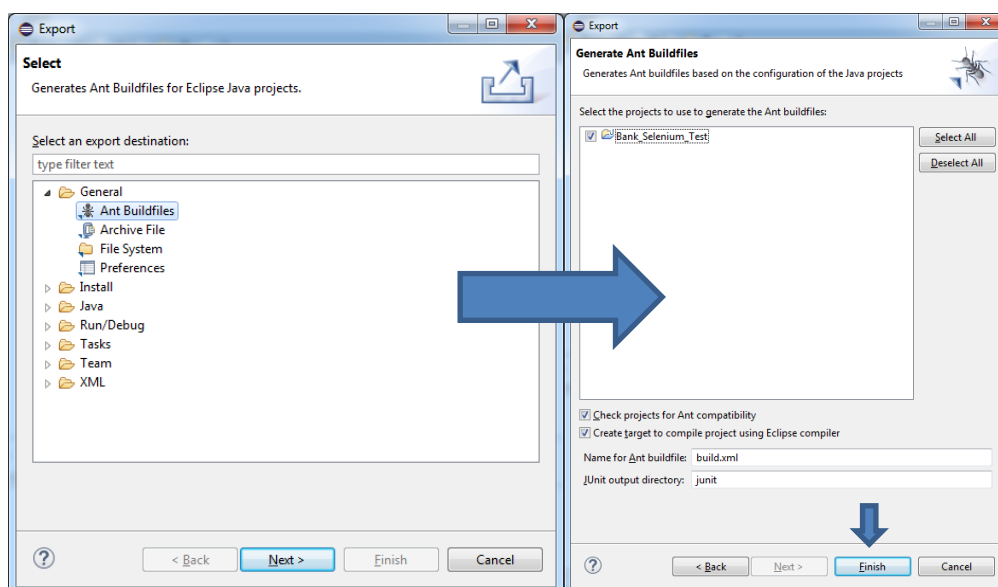


Figura 21 – Geração do arquivo build.xml

Feito isso, selecionamos dentro de “General” a opção “Ant Builders” e clicamos em “Next”. Na tela seguinte verificamos se as configurações do projeto, nome do arquivo de “build” estão correto e clicamos em “Finish”. Desse modo, surgirá no projeto um arquivo com o nome “build.xml”.

Tendo gerado o arquivo de “build” com sucesso o nosso projeto já está pronto para ser utilizado na integração contínua do Jenkins, porém antes de avançarmos para esse passo devemos testar localmente se a *build* irá testar a nossa aplicação corretamente.

Para executarmos a *build* de modo semelhante a qual o Jenkins fará posteriormente clicamos com o botão direito no arquivo de “build” recém-gerado e selecionamos “Run As -> Ant Build”. Em seguida visualizaremos a seguinte tela:

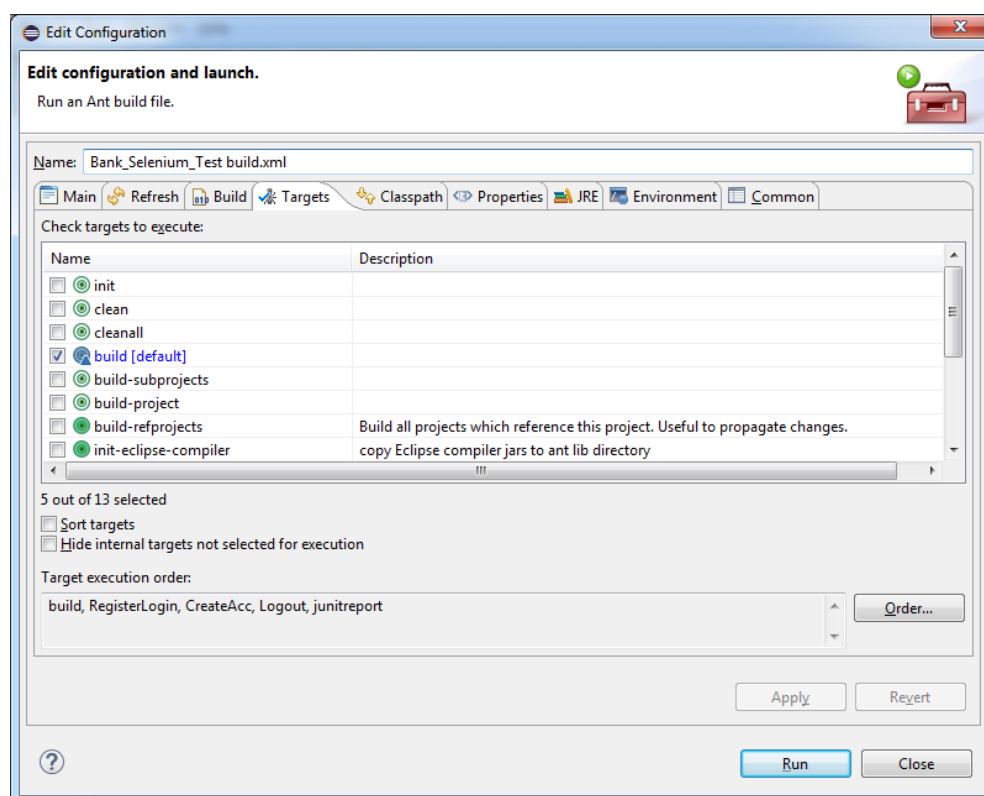
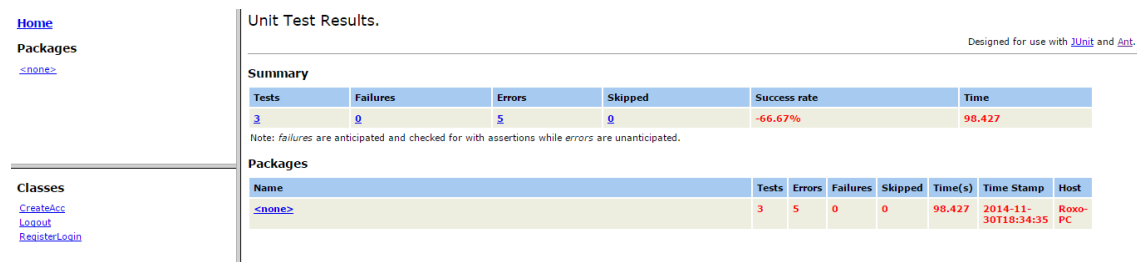


Figura 22 – Configuração da execução do arquivo build.xml

Nessa janela é possível selecionar os métodos Ant que desejamos executar durante a construção do projeto. Selecionamos as seguintes opções: “build”, todos os casos de testes e a opção “junit report”. No campo “Target execution order” ordenamos

de maneira que a “build” seja feita antes dos testes e o “junitreport” seja feito após. Feito isso, clicamos em “Run” e aguardamos enquanto todos os nossos casos de teste são executados no navegador Firefox automaticamente.

Ao fim dos testes, o JUnit irá criar uma pasta “junit” dentro do nosso projeto com todos os resultados em forma de paginas HTML. Ao abrirmos a página “index.html” podemos verificar se os resultados dos testes foram de acordo com o esperado.



The screenshot shows the JUnit Test Results page. On the left, there are links for Home, Packages, and Classes. The main content area is titled 'Unit Test Results.' and includes a summary table and a packages table. The summary table shows 3 tests, 0 failures, 5 errors, 0 skipped, a success rate of -66.67%, and a time of 98.427. The packages table shows one package named '<none>' with 3 tests, 5 errors, 0 failures, 0 skipped, a time of 98.427, a timestamp of 2014-11-30T18:34:35, and a host of Roxo-PC.

Unit Test Results.						
Summary						
Tests	Failures	Errors	Skipped	Success rate	Time	
3	0	5	0	-66.67%	98.427	

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Packages							
Name	Tests	Errors	Failures	Skipped	Time(s)	Time Stamp	Host
<none>	3	5	0	0	98.427	2014-11-30T18:34:35	Roxo-PC

Figura 23 – Resultados dos testes

5.4. Integração Contínua com Jenkins

Com toda a solução de testes construída e o arquivo de construção (build.xml) configurado com o Ant (de modo que os testes podem ser executados durante a construção), precisamos apenas configurar o Jenkins para iniciar o ciclo de integração contínua no projeto.

Primeiramente, serão necessários alguns plugins do Jenkins. No menu de gerência de plugins deve ser efetuado o download dos seguintes plugins:

- GitHub Plugin;
- Ant Plugin;
- JUnit Plugin;

Após o download e a instalação dos plugins, devemos configurar os caminhos do JDK e do Ant nas configurações de sistema do Jenkins, como pode ser visto abaixo:

JDK

JDK instalações

JDK
Nome: Java_Home

JAVA_HOME: C:\Program Files\Java\jdk1.8.0_25

☐ Instalar automaticamente

Ant

Ant instalações

Ant
Nome: Ant

ANT_HOME: C:\Program Files (x86)\Eclipse\plugins\org.apache.ant_1.9.2.v201404171502

☐ Instalar automaticamente

Figura 24 – Configurações dos caminhos do JDK e do Ant

Além disso, será necessário que configuremos, na mesma página de configuração de sistema, o servidor SMTP de envio de e-mails. No caso desse projeto utilizamos o serviço do gmail para realizar esse envio, como pode ser visto abaixo:

Notificação de E-mail

Servidor SMTP: smtp.gmail.com

Sufixo padrão para e-mail de usuário: @gmail.com

☒ Usar autenticação SMTP

Nome do Usuário: leonardoroxo@poli.ufrj.br

Senha:

☒ Usar SSL

Porta SMTP: 465

Reply-To Address:

Charset: UTF-8

☐ Configuração de teste para enviar e-mail

Figura 25 – Configuração do servidor SMTP

Feito isso, o Jenkins já está com seu sistema devidamente configurado. Para completarmos o ciclo de integração basta que configuremos ambos os projetos como “Novos Jobs”.

Para isso criamos um primeiro “Job” como um projeto de estilo livre no Jenkins e o configuramos para construir “builds” do projeto principal “Online Bank”. Nesse caso, foi utilizado o GitHub como repositório para esse projeto.

Trigger de builds

☒ Build when a change is pushed to GitHub

Figura 26 – Trigger do GitHub

Marcamos a opção acima para tornar automática a geração de “builds” sempre que o desenvolvedor realizar um “commit” de um novo código.

Por fim um segundo “Job” é configurado. Este será responsável pelo projeto de testes que criamos anteriormente.

O segundo “Job” também é criado como projeto de estilo livre e configurado para construir “builds” do projeto de testes “Bank Selenium Test”.

Trigger de builds

☐ Build when a change is pushed to GitHub

☒ Construir após a construção de outros projetos

Projetos observados:

☒ Disparar apenas se o build estiver estável

☐ Disparar mesmo se o build estiver instável

☐ Disparar mesmo se o build falhar

Figura 27 – Configuração do Trigger

Como o projeto de testes deve ser executado logo após cada construção do projeto “Online Bank”, utilizamos o trigger descrito na figura 27 para acionar a execução desse projeto sempre que houver “build” estável do projeto principal.

Build

Invocar Ant

Versão do Ant:

Alvos:

 Avançado...

Figura 28 – Configuração do Ant

Além disso, devemos acionar todos os métodos Ant das classes de teste geradas para serem executadas durante a construção do “build” do projeto de testes, como descrito na figura 28.

Ações de pós-build

Publicar relatório de testes do JUnit ?

Relatório XML de teste

Descrição

☐ Manter padrão de erro de saída ?

Health report amplification factor ?

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Figura 29 - Configuração do Relatório de JUnit

São definidos também a origem e quais os arquivos que o relatório dos testes do JUnit utilizará para reportar os resultados dos testes. Isso pode ser observado na figura 29.

Notificação de E-mail ?

Destinatários

Lista de endereços separados por espaços em branco. E-mail será enviado quando uma construção falhar.

☒ Enviar e-mail para todas as construções instáveis

☐ Enviar e-mails separados para os indivíduos que danificaram a construção ?

Figura 30 – Configuração de e-mail

Por fim, configuramos a notificação de e-mail descrita na figura 30, na qual é definida quem deve ser acionado em caso de falhas na construção do projeto.

5.5. Manutenção do ciclo de Integração Contínua (CI)

A manutenção do ciclo deve ser e foi construída de modo que ambos os projetos (teste e desenvolvimento) possam evoluir sem que um atrapalhe o outro. Como vimos, a ideia principal da automatização é reduzir esforços, logo seria inaceitável que nosso projeto tivesse um gargalo baseado na comunicação desenvolvedor-*tester*.

Os desenvolvedores terão total liberdade para avançar na codificação, tendo sempre em mente que os testes são baseados na interface, ou seja, qualquer modificação que afete isso resultará em uma versão falha do projeto.

Da mesma maneira os *testers* podem melhorar os casos de testes existentes ou criar mais casos testando-os sempre na versão mais atual do código.

Capítulo 6

Resultados e Conclusões

Neste capítulo, apresentaremos uma análise dos resultados obtidos com este estudo sobre automação de testes na área de software e as conclusões tiradas baseadas nesses resultados.

6.1. Resultados

Analisando todo o estudo que foi feito e o resultado atingido com a construção da integração contínua para desenvolvimento e teste do software “Online Bank”, conclui-se que este projeto de graduação atingiu o resultado esperado.

Através desse estudo foi possível chegar a um melhor entendimento dos conceitos de testes, suas nomenclaturas e principais modelos.

Além disso, adquirimos conhecimento sobre os diversos tipos de testes e seus principais casos de aplicação. Somado a isso, entendemos experimentalmente, durante o caso de estudo que nem todos os testes se aplicam em todas as situações. Isso pode ser percebido ao tentar inserir testes unitários em um código legado já em uso.

Ao longo do estudo de caso, conseguimos aprender sobre diversas ferramentas de automação de software, entendendo seu funcionamento e configurações.

Por fim, analisando a integração que foi construída integrando o teste e o desenvolvimento, acreditamos que o objetivo principal desse estudo foi obtido.

6.2. Conclusão

Através do estudo apresentado por esse projeto foi possível perceber que em qualquer software, do mais simples ao mais complicado, sempre é necessário que sejam definidos e executados testes antes da entrega para o usuário final. Tal afirmativa se baseia nos princípios de qualidade que todo usuário deseja ter ao utilizar qualquer ferramenta.

Somado a isso, percebemos que a premissa de que todo teste deve ser automatizado se mostrou inválida. Automatizar tem grande valor na maioria dos casos, porém existem situações que sua inserção encarece o projeto além de requisitar uma manutenção longa e constante. Dessa forma, nesses casos os testes manuais ainda são mais efetivos e econômicos.

Dentro do universo de automação, fomos capazes de perceber que os testes de regressão passam a ter um valor mais significativo. Ao automatizar esse teste, o projeto passa a ter uma proteção contra a inserção de novos defeitos que alerta prontamente ao desenvolvedor.

Quanto ao estudo de caso, concluímos que teve imenso valor de aprendizado. Apesar de o sistema bancário utilizado ser extremamente simples, o que facilitaria a resolução de qualquer defeito que encontramos durante os testes, ele serviu como um exemplo prático representando um software mais complexo em uma situação real. E mesmo sendo simples já foi possível perceber a dificuldade que existe em se aplicar o conceito de teste unitário em projetos que já estão em andamento com código legado.

Ao fim do estudo de caso, percebemos que o fato de utilizarmos o servidor Apache TomCat vinculado ao Netbeans foi um facilitador. Porém em uma situação mais real isso terá que ser automatizado também através do Jenkins.

Referências

[1] Artigo Engenharia de Software - Introdução a Teste de Software, disponível em <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>(acessado em 23/11/2014).

[2] Understanding White box Testing and Black box Testing Approaches, disponível em <http://blog.testing-whiz.com/2011/11/understanding-white-box-testing-and.html>(acessado em 23/11/2014).

[3] Relationship between Servlet and JSP, disponível em <http://www.herongyang.com/JSP/Servlet-Relationship-between-Servlet-and-JSP.html>(acessado em 26/11/2014).

[4]STERLING, B. The Hacker Crackdown: Law and Disorder on the Electronic Frontier. Bantam Books, 1992

[5]BOEHM, BARRY W. and PHILIP N. PAPACIO. Understanding and Controlling Software Costs, IEEE Transactions on Software Engineering, v. 14, no. 10, October 1988, pp. 1462-1477.

[6]FOWLER, MARTIN. Continuous Integration, <http://martinfowler.com/articles/originalContinuousIntegration.html> (acessado em 23/11/2014).

Bibliografia

Artigo Engenharia de Software - Introdução a Teste de Software,
<http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035> (Acessado em 23/11/2014).

Understanding White box Testing and Black box Testing Approaches,
<http://blog.testing-whiz.com/2011/11/understanding-white-box-testing-and.html>
(Acessado em 23/11/2014).

Testes de unidade com JUnit,
<http://www.devmedia.com.br/testes-de-unidade-com-junit/4637> (Acessado em 23/11/2014).

Testes de Integração com Java e JUnit,
<http://www.devmedia.com.br/testes-de-integracao-com-java-e-junit/25662> (Acessado em 23/11/2014).

JUnit - Implementando testes unitários em Java – Parte I,
<http://www.devmedia.com.br/junit-implementando-testes-unitarios-em-java-parte-i/1432> (Acessado em 23/11/2014).

Understanding White box Testing and Black box Testing Approaches,
<http://blog.testing-whiz.com/2011/11/understanding-white-box-testing-and.html>
(Acessado em 23/11/2014).

TechZoo,
<http://www.techzoo.org/projects/online-bank-management-system-project-in-java.html>
(Acessado em 23/11/2014).

Apêndice A

Casos de Teste

<u>Caso de Teste:</u> 001 – Criar Usuário		
<u>Objetivo:</u> Validar a criação de novos usuários.		
<u>Precondições:</u> <ul style="list-style-type: none">• Ter acesso à página inicial do site “Online Bank” (http://localhost:8084/Bank);		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>
1	Clicar na opção “Click Here” logo após a mensagem “Want to create a new account?”.	Sistema deve navegar até a página para registro de novo usuário.
2	Preencha os campos: “Username”, “Password”, “Address”, “E-mail” e “Mobile” com valores válidos. Escolha uma pergunta secreta em “Security Question” e preencha uma resposta em “Answer”. Clique em “Reset”.	Todos os campos devem ser apagados.
3	Preencha os campos: “Username”, “Password”, “Address”, “E-mail” e “Mobile” com valores válidos. Escolha uma pergunta secreta em “Security Question” e preencha uma resposta em “Answer”.	Sistema navega para uma tela com a confirmação de criação de usuário exibindo a seguinte mensagem: “Account is Created Successfully. <u>Click Here</u> to Login and Acticvate Your Account.”.

	Clique em “Create Account”.	
4	Clique em “Click Here”.	Sistema redireciona para a tela inicial de Login.
5	Preencha o Login com o “Username” recém-criado e seu respectivo “Password”. Clique em “Submit”.	Sistema autentica o usuário e mostra a tela inicial da conta do usuário.
6	Clique em “LogOut”.	Sistema redireciona para a tela inicial de Login.
<u>Resultados:</u>		
<u>Comentários:</u>		

<u>Caso de Teste:</u> 002 – Navegabilidade sem conta bancária		
<u>Objetivo:</u> Validar navegabilidade no site sem possuir nenhuma conta bancária.		
<u>Precondições:</u> <ul style="list-style-type: none"> Ter efetuado “Login” com um usuário que não possua contas no banco; 		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>
1	Clique na opção “Deposit”.	Sistema deve navegar para uma nova tela informando que o usuário não possui conta bancária com a seguinte mensagem: “Your do not have any account created. To create Your Account <u>Click Here</u> . Or to Log out <u>Click Here</u> ”.
2	Clique na opção “DoWithdraw”.	Sistema deve navegar para uma nova tela informando que o usuário não

		<p>possui conta bancária com a seguinte mensagem: “Your do not have any account created.</p> <p>To create Your Account Click Here.</p> <p>Or to Log out Click Here”.</p>
3	Clique na opção “Get Balance”.	<p>Sistema deve navegar para uma nova tela informando que o usuário não possui conta bancária com a seguinte mensagem: “Your do not have any account created.</p> <p>To create Your Account Click Here.</p> <p>Or to Log out Click Here”.</p>
4	Clique na opção “Transfer Amount”.	<p>Sistema deve navegar para uma nova tela informando que o usuário não possui conta bancária com a seguinte mensagem: “Your do not have any account created.</p> <p>To create Your Account Click Here.</p> <p>Or to Log out Click Here”.</p>
5	Clique na opção “View Report”.	<p>Sistema deve navegar para uma nova tela informando que o usuário não possui conta bancária com a seguinte mensagem: “Your do not have any account created.</p> <p>To create Your Account Click Here.</p> <p>Or to Log out Click Here”.</p>
6	Clique na opção “Click Here” que efetuará o Log out.	Sistema deve ser redirecionado para tela de Login.
7	Efetue novamente o Login com o mesmo usuário.	Sistema deve navegar para uma nova tela informando que o usuário não possui conta bancária com a seguinte

	Clique novamente na opção “View Report”.	mensagem: “Your do not have any account created. To create Your Account Click Here . Or to Log out Click Here ”.
8	Clique na opção “Click Here” para criar nova conta bancária.	Sistema navega para a tela “Create Account”.
<u>Resultados:</u>		
<u>Comentários:</u>		

<u>Caso de Teste:</u> 003 – Criar conta bancária		
<u>Objetivo:</u> Validar a criação de conta bancária.		
<u>Precondições:</u> <ul style="list-style-type: none"> Ter efetuado “Login” com um usuário; 		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>
1	Clique na opção “Create Account”.	Sistema navega para a tela “Create Account” com os campos “Account Holder Name” e “Account Number” previamente preenchidos e desabilitados para edição.
2	Selecione um tipo de conta em “Account Type” e escreva algum comentário em “Account Details”. Clique em “Create Account”	A conta é criada com sucesso e o Sistema navega até uma página de confirmação com a seguinte mensagem: “Your account is successfully created. and Account No. is <Account Number>. To Deposit Amount in Your Account Click Here To Withdraw From Account Click

	<u>Here</u> ".
<u>Resultados:</u>	
<u>Comentários:</u>	

<u>Caso de Teste:</u> 004 – Depositar		
<u>Objetivo:</u> Validar depósitos na conta bancária.		
<u>Precondições:</u> <ul style="list-style-type: none"> • Ter efetuado “Login” com um usuário; 		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>
1	Clique em “Deposit”	Sistema navega para a tela “Deposit” com o campo “Account Holder Name” previamente preenchido e desabilitado para edição.
2	Selecione o número de uma conta em “Account Number” e defina o valor que será depositado no campo “Deposit Amount” Clique no botão “Deposit Amount”.	O depósito é efetuado com sucesso e o Sistema navega até uma página de confirmação com a seguinte mensagem: “Your Amount is Successfully Deposited into Account. Your current Balance is: <\$\$> Rs. To Deposit Amount in Your Account <u>Click Here</u> To Withdraw From Account <u>Click Here</u> ”.
<u>Resultados:</u>		
<u>Comentários:</u>		

<u>Caso de Teste:</u> 005 – Saques		
<u>Objetivo:</u> Validar saques na conta bancária.		
<u>Precondições:</u> <ul style="list-style-type: none"> Ter efetuado “Login” com um usuário; 		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>
1	Clique em “Do Withdraw”	Sistema navega para a tela “Withdraw” com o campo “Account Holder Name” previamente preenchido e desabilitado para edição.
2	Selecione o número de uma conta em “Account Number” e defina o valor (que exista na conta) que será sacado no campo “Withdraw Amount” Clique no botão “Withdraw Amount”.	O saque é efetuado com sucesso e o Sistema navega até uma página de confirmação com a seguinte mensagem: “Your Amount is Successfully Withdraw from Account. To Deposit Amount in Your Account Click Here To Withdraw From Account Click Here ”.
<u>Resultados:</u>		
<u>Comentários:</u>		

<u>Caso de Teste:</u> 006 – Balanço		
<u>Objetivo:</u> Validar visualização de balanço da conta bancária.		
<u>Precondições:</u> <ul style="list-style-type: none"> Ter efetuado “Login” com um usuário; 		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>

1	Clique em “Get Balance”	Sistema navega para a tela “Get Balance” com o campo “Account Holder Name” previamente preenchido e desabilitado para edição.
2	<p>Selecione o número de uma conta em “Select Account No.”.</p> <p>Clique no botão “Check Balance”.</p>	<p>O sistema atualiza a página exibindo o saldo atual da conta no campo “Current Balance”, seguido pela seguinte mensagem: “Your Amount is Successfully Withdraw from Account. To Deposit Amount in Your Account Click Here To Withdraw From Account Click Here”.</p>
<u>Resultados:</u>		
<u>Comentários:</u>		

<u>Caso de Teste:</u> 007 – Transferências		
<u>Objetivo:</u> Validar transferências de conta bancária.		
<u>Precondições:</u> <ul style="list-style-type: none"> • Ter efetuado “Login” com um usuário; • Existirem ao menos 2 contas bancárias no sistema; 		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>
1	Clique em “Transfer Amount”	Sistema navega para a tela “Transfer Amount” com o campo “Account Holder Name” previamente preenchido e desabilitado para edição.
2	Selecione o número de uma	A transferência é efetuada com sucesso

	<p>conta origem da transferência em “Select Source Account No.”, selecione o número de uma conta destino da transferência em “Select Destination Account No.”, defina o valor (disponível na conta) que será transferido no campo “Amount” e defina algum comentário para a transação em “Details”.Clique no botão “Transfer Amount”.</p>	<p>e o Sistema navega até uma página de confirmação com a seguinte mensagem: “Your Amount <\$\$> is Successfully Transfer to Acount <Account Number>. To Deposit Amount in Your Account Click Here To Withdraw From Account Click Here”.</p>
Resultados:		
Comentários:		

<u>Caso de Teste:</u> 008 – Relatório		
<u>Objetivo:</u> Validar geração de relatório de transações.		
<u>Precondições:</u> <ul style="list-style-type: none"> • Ter efetuado “Login” com um usuário; • Ter ao menos uma conta com esse usuário com ao menos uma transação realizada; 		
<u>Step:</u>	<u>Ação:</u>	<u>Resultado esperado:</u>
1	Clique em “View Report”	O Sistema exibe uma tela com todas as transações já executadas exibindo as colunas: “Acc No.”, “Operation”, “Amt”, “Balance” e “Date – Time”.
<u>Resultados:</u>		
<u>Comentários:</u>		

