

**FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO
NORTE**

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

BRUNO VINÍCIUS MOURA DOS REIS

**ESTUDO E APLICAÇÃO DE TÉCNICAS EM AUTOMAÇÃO DE TESTES
FUNCIONAIS DE SOFTWARE**

NATAL, RIO GRANDE DO NORTE.

2009

BRUNO VINÍCIUS MOURA DOS REIS

**ESTUDO E APLICAÇÃO DE TÉCNICAS EM AUTOMAÇÃO DE TESTES
FUNCIONAIS DE SOFTWARE**

Monografia apresentada como exigência
do Curso de Sistemas de Informação da
Faculdade Natalense para o
Desenvolvimento do Rio Grande do
Norte para obtenção do título de
Bacharel.

Orientador: MsC. Ricardo Wendell
Rodrigues da Silveira.

NATAL, RIO GRANDE DO NORTE.

2009.

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Reis, Bruno Vinicius Moura dos

Estudo e aplicações técnicas em automação de testes funcionais de software. / Bruno Vinicius Moura dos Reis. – Natal, RN, 2009.
49f.

Orientador: Ricardo Wendell Rodrigues da Silveira (Msc.)

Monografia (Bacharel em Sistema de Informação) – Faculdade Natalense
para o Desenvolvimento do Rio Grande do Norte.

1. Automação - Monografia. 2. Programação de Software - Monografia.

I. Título.

RN/FARN/BC

CDU 004.43

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO
NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

BRUNO VINÍCIUS MOURA DOS REIS

ESTUDO E APLICAÇÃO DE TÉCNICAS EM AUTOMAÇÃO DE TESTES
FUNCIONAIS DE SOFTWARE

Monografia apresentada e aprovada em 14 de Dezembro de 2009, pela banca examinadora
composta pelos seguintes membros



Msc. Ricardo Wendell Rodrigues da Silveira

Orientador

Msc. Raphaela Galhardo Fernandes

Membro Examinador

Msc. Gleydson de A. F. Lima

Membro Examinador

Natal/RN, 14 de Dezembro de 2009.

Agradecimentos

A minha mãe, fonte de todo o meu conhecimento e inspiração, que mesmo não estando ao meu lado nos momentos de conclusão do meu curso sei que de alguma forma está participando da felicidade de está cumprindo uma etapa da minha vida, a mais esperada por ela, e a dedico essa mensagem para expressar todo meu agradecimento por tanto cuidado e dedicação durante os vinte e dois anos que estive ao meu lado:

Guiou meus passos com muito amor,
Esta presente em meu ser,
A vida inteira se dedicou,
Com luta e fé não desanimou.

Mãe um dia vou te encontrar
E nos seus braços vou estar
Para entregar o que aqui Deus me guardou
E suas sementes farei brotar.

Nesse lugar que a terra seca
Não vai me derrotar
Por que esse amor que em mim gerou
Foi a maior parte que de ti ficou.

Ao meu tio Aluizio de Andrade Moura Sobrinho, peça fundamental nessa conquista, pelo apoio e incentivo dado durante esses quatro anos.

Ao meu orientador, pela disposição e comprometimento na orientação do meu trabalho.

RESUMO

Este trabalho propõe o estudo e aplicação de técnicas em automação de testes funcionais de software, implantando um processo de automação utilizando a ferramenta TestComplete em uma *Software House*, analisando ao final os resultados obtidos.

Essas técnicas consistem em desenvolver scripts de teste que promovam uma independência e isolamento entre eles, aumentando a eficiência na execução e o reuso de código, proporcionando uma fácil manutenibilidade dos scripts de teste.

Palavras-Chave: Teste de Software. Automação de testes. Técnicas de automação de testes.

ABSTRACT

This work proposes the study and application of techniques in functional testing automation of software, implementing the automation process using the *TestComplete* tool in a *Software House*, and thus analyzing the results.

These techniques are used to develop test scripts that promote independence and isolation among them, increasing efficiency in the implementation and reuse of code, providing an easy maintainability of test scripts.

Key-words: Software Testing. Test Automation. Automation systems testing.

SUMÁRIO

Índice de Figuras	viii
Índice de Tabelas	ix
1. INTRODUÇÃO	10
1.1. Objetivos	11
1.2. Justificativa.....	11
1.3. Organização do texto.....	12
2. REFERENCIAL TEÓRICO	13
2.1. Teste de Software	13
2.2. Automação de Testes Funcionais	17
2.2.1. Técnicas de automação de teste.....	18
2.2.1.1. Record & Playback	18
2.2.1.2. Programação de Scripts.....	18
2.2.1.3. <i>Data-Driven</i> (Técnica Orientada a Dados).....	19
2.2.1.4. <i>Keyword-Driven</i> (Técnica Orientada a Palavras-Chaves)	19
2.2.1.5. Decomposição Funcional.....	20
2.4. Estratégias de sucesso na automação	23
2.5. Dificuldades na automação de testes.....	26
3. DESENVOLVIMENTO	27
3.1. Identificação de Casos de Teste	29
3.2. Construção.....	29
3.3. Execução	34
3.1. Validação.....	38
4. RESULTADOS OBTIDOS	39
4.1. Contexto	39
4.1.1 Escopo do Processo de Automação	39

4.2.	Análise dos resultados	39
4.2.1	Custo da Automação.....	40
4.2.2	Custo dos Testes Manuais	42
4.2.3	Melhoria no Tempo de Execução dos Testes	44
4.2.4	Melhoria no Custo de Execução dos Testes	44
4.2.5	Retorno do Investimento.....	44
5.	CONCLUSÕES E PERSPECTIVAS	46
5.1.	Dificuldades Encontradas.....	46
5.2.	Trabalhos Futuros.....	47
6.	BIBLIOGRAFIA	48

Índice de Figuras

<i>Figura 1 – Fluxo de atividades do projeto de automação</i>	<i>25</i>
<i>Figura 2 – Arquitetura do projeto de automação</i>	<i>27</i>
<i>Figura 3 – Scripts utilizados pela arquitetura</i>	<i>29</i>
<i>Figura 4 – AutomatedQA QATime.....</i>	<i>34</i>
<i>Figura 5 – AutomatedQA TestComplete</i>	<i>34</i>
<i>Figura 6 – Árvore dos Stores.....</i>	<i>36</i>
<i>Figura 7 – Lista de Checkpoint.....</i>	<i>36</i>

Índice de Tabelas

<i>Tabela 1</i> – Fonte Bartié (2002, p.64).....	16
<i>Tabela 2</i> – Custo das atividades	39
<i>Tabela 3</i> – Custos fixos da automação	41
<i>Tabela 4</i> – Custos variáveis da automação.....	41
<i>Tabela 5</i> – N° de campos de cadastros	42
<i>Tabela 6</i> – Redução de custo da execução	43
<i>Tabela 7</i> – Retorno do investimento	43

1. INTRODUÇÃO

Nos dias de hoje, com a crescente concorrência entre as empresas de software e o aumento no nível de exigência de seus usuários, tem-se exigido um processo de teste de software cada vez mais aprimorado e bem definido. Desta forma, essas empresas vêm constantemente buscando alternativas no processo de teste para revelar erros na fase de desenvolvimento, reduzindo custos na sua correção e agregando qualidade ao produto final.

Dependendo do nível de complexidade do software a dificuldade aumenta e torna-se trabalhoso testar todo o sistema para liberar uma nova versão. Diante disto empresas têm visto a necessidade de investir em automação de teste com o uso de ferramentas, visando uma verificação mais rápida e eficiente das falhas no software e não conformidades com os requisitos expressados pelos clientes, em alguns casos essa verificação é exigida nos modelos de referência em processo de desenvolvimento de software como CMMI e MPS-BR. Entre essas ferramentas podemos destacar alguns exemplos como o *Rational Visual Test*, *TestComplete*, *Borland SilkPerformer Component Test* e *Selenium*, sendo este último de código-aberto.

A automatização dos testes é altamente desejada por diversos fatores, inclusive em termos de custos finais. [...] À medida que reexecutamos os testes, o ganho de tempo, controle, confiabilidade e as diversas possibilidades existentes com essa tecnologia, fica clara a vantagem inerente a esse processo. (BARTIÉ, 2002, p. 197).

Entretanto, para implantar uma solução de testes automatizados se faz necessário um alto investimento inicial da empresa: recursos humanos, dinheiro e tempo. É preciso antes de pensar na automação ter um processo de teste bem definido, analisar as expectativas e benefícios a ser alcançado com a automação, já que no início é dispensado um tempo significativo no desenvolvimento dos *scripts de testes*, passos gerados em uma linguagem de programação das ações executadas na tela do computador, há casos que se gasta mais tempo em desenvolver os testes do que a própria funcionalidade em si. Além de softwares sofrerem sempre alterações, e essas poderem refletir em uma necessidade de reprogramação em todos os *scripts* para que funcionem corretamente.

Diante do exposto serão tratadas neste trabalho algumas dessas problemáticas envolvidas na automação, buscando estudar e aplicar técnicas de automação de testes em uma empresa de desenvolvimento de software real, a fim de colher e analisar os resultados e propor boas práticas a serem consideradas em um processo de automação de teste funcional.

1.1. Objetivos

Este trabalho tem como principal objetivo o estudo e aplicação de técnicas de automação de teste de software em uma fábrica de software, analisar os resultados obtidos, relatar os pontos positivos e negativos do estudo proposto e apresentar boas práticas para auxiliar em futuras implantações de um processo de automação de teste. E tem como objetivos específicos:

- Analisar as técnicas de automação de testes disponíveis;
- Estudar uma ferramenta de automação para a aplicação dessas técnicas;
- Elaborar projeto de automação de teste funcional;
- Aplicar técnicas de automação de testes em uma empresa de desenvolvimento de software para automação comercial;
- Relatar os resultados obtidos com o estudo realizado.
- Realizar uma análise comparativa dos resultados obtidos em cada técnica;
- Documentar os resultados do estudo de caso.

1.2. Justificativa

Teste de software sendo uma ferramenta do processo de qualidade que auxilia na verificação do produto, é uma atividade não praticada corretamente por muitas empresas de desenvolvimento que acabam não se organizando nas outras fases do projeto e sacrificam a fase de testes para cumprir os prazos acordados com os clientes entregando produtos com baixa qualidade e muitos retornos para correção.

O estudo que será realizado com este trabalho se justifica com os resultados que serão obtidos e que poderão servir como auxílio na decisão para implantação de um processo de

automação de teste, ajudando em uma verificação mais rápida do produto e encontrando erros cada vez mais cedo diminuindo os custos na sua correção, serão analisados os pontos positivos e negativos e o ganho real obtido em custos e qualidade na aplicação do estudo em uma *Software House* no ramo de automação comercial.

1.3. Organização do texto

O texto desta monografia está organizado em cinco capítulos, nos quais está desenvolvido o tema. No primeiro capítulo foi apresentada uma introdução e motivação do trabalho. No capítulo 2 será exposto o referencial teórico referente aos estudos e estratégias existentes para a automação em testes funcionais de software. No cap. 3 apresentaremos uma explanação de como as técnicas explicadas no capítulo anterior foram aplicadas. No cap. 4 serão apresentados os resultados obtidos no estudo de caso que se encerra no cap. 5 com suas conclusões com as dificuldades encontradas e sugestões para continuidade do estudo em trabalhos futuros.

2. REFERENCIAL TEÓRICO

Neste capítulo será apresentado o referencial teórico necessário ao entendimento do tema exposto neste trabalho.

2.1. Teste de Software

De acordo com a norma IEEE (2004), teste de software é o processo em que um sistema ou componente é executado sob condições específicas. Os resultados são observados, registrados e uma avaliação é realizada sobre os resultados, tendo como principal objetivo revelar os defeitos de um programa. Conforme define Myers (2004), “teste é o processo de executar o programa com a intenção de achar erros”.

Sendo assim a atividade de teste tem grande importância na fase de desenvolvimento do software a fim de encontrar erros que foram introduzidos durante todo o processo, desde a análise até o desenvolvimento.

Diferentes tipos de testes podem ser realizados com diversos objetivos. Dentre eles, temos: funcionais, performance, carga, volume, instalação, integridade, estresse, usabilidade, regressão e sanidade.

Testes funcionais são baseados na especificação do software ou componente, existe uma abordagem de caixa preta que segundo Pressman (2006, p. 816), são “métodos de testes que se concentram nos requisitos funcionais do software, ele tem como objetivo descobrir erros nas seguintes categorias: (1) funções incorretas ou ausentes; (2) erros de interface; (3) erros nas estruturas de dados ou no acesso a banco de dados externos; (4) erros de desempenho; e (5) erros de inicialização e término”, esse tipo de teste é aplicado por um testador, focado somente nas funcionalidades do sistema e não com a implementação, e devem verificar se o software está atendendo os requisitos expressos pelos clientes e especificados pelos engenheiros na fase de análise.

Existe outra abordagem que são os testes de caixa branca, são testes estruturais, realizados na estrutura interna do código, que Pressman (2006, p. 793 e 794), define como um método de projeto de casos de teste que usa a estrutura de controle do projeto procedimental para derivar casos de testes, possibilita que engenheiro de software derive os casos de testes que (1) garantam que todos os caminhos independentes dentro de um módulo tenham sido

exercitados pelo menos uma vez; (2) exercitem todas as decisões lógicas para valores falsos ou verdadeiros; (3) executem todos os laços em suas fronteiras e dentro de seus limites operacionais; e (4) exercitem as estruturas de dados internas para garantir a sua validade.

O teste de caixa preta não é uma alternativa para as técnicas de caixa branca, ao contrário, trata-se de uma abordagem complementar que tem a probabilidade de descobrir uma classe de erros diferentes dos que são encontrados nos testes de caixa branca, e que se aplicadas em conjunto podem oferecer bons níveis de qualidade ao produto. (PRESSMAN 2006)

Esses testes e análise são voltados para verificação, isto é, voltado para encontrar discrepâncias entre o que um programa faz e o que deveria fazer – deve-se ter como base as informações dos requisitos expressos por usuários e especificados pelos engenheiros de software. Que produzirão especificações funcionais, ou seja, uma descrição do comportamento esperado do programa, essa é a principal fonte de informação para a elaboração dos casos de testes, esses expressão como o software deve se comportar diante de uma especificação funcional. (Pezzé e Young 2008, p. 182)

A derivação de casos de teste funcional é um processo analítico realizado pelo projetista de teste, com a finalidade de decompor e estruturar adequadamente as especificações funcionais em casos de teste, escolhendo os critérios de seleção de teste, que definem a estratégia de geração de especificações de casos de teste.

Algumas técnicas e métodos podem ser aplicados na geração dos casos de testes MOLINARI (2003, p. 168):

Particionamento de Equivalência

Divide o domínio de entrada de um programa em classes de dados das quais os casos de testes podem ser derivados. Tem a finalidade de descobrir classes de erros, assim reduzindo o número total de casos de testes que devem ser resolvidos e podem ser definidos de acordo com as seguintes diretrizes:

1. Se uma condição de entrada especificar um *intervalo*, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas.

2. Se uma condição de entrada exigir um *valor* específico, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas.
3. Se uma condição de entrada especificar um membro de um *conjunto*, uma classe de equivalência válida e uma classe de equivalência inválida são definidas.
4. Se uma condição de entradas for *booleana*, uma classe válida e uma inválida são definidas.

Análise do Valor Limite

Um número maior de erros tende a ocorrer mais nas fronteiras do domínio de entrada do que no “centro”, essa técnica põe a prova os valores fronteiros complementando o particionamento de equivalência, e suas diretrizes se assemelham às apresentadas para o particionamento de equivalência:

1. Se uma condição de entrada especificar um intervalo, delimitado pelos valores *a* e *b*, os casos de testes devem ser projetados com valores *a* e *b* logo acima e logo abaixo de *a* e *b* respectivamente.
2. Se uma condição de entrada receber uma série de valores, deve-se testar o valor máximo e o mínimo da série. Valores logo acima e logo abaixo do mínimo e do máximo também são testados.
3. Aplica-se as diretrizes 1 e 2 às condições de saída. Exemplo: se em um programa de engenharia fossem exigidos uma tabela com os valores da relação entre temperatura e pressão como saída. Os casos de teste devem ser projetados para criar relatórios de saída que produzam um número máximo (e mínimo) permissível de entradas na tabela.
4. Se estruturas internas de dados do programa tiverem prescrito fronteiras, deve-se projetar um caso de teste para exercitar a estrutura de dados em sua fronteira.

Técnicas de Grafo de Causa e Efeito

É uma técnica de projeto de caso de teste que oferece uma representação concisa das condições lógicas e das ações correspondentes, com seguintes passos:

1. Causas (condições de entrada) e efeitos (ações) são relacionados para um módulo e um identificador é atribuído a cada um.
2. Um grafo de causa efeitos é desenvolvido.
3. O grafo é convertido numa tabela de decisão.
4. As regras da tabela de decisão são convertidas em casos de teste.

Keith (1998, p. 4), mostra que as formas de aplicação desses testes funcionais podem ser manuais, onde uma pessoa executa o teste previamente definido e documentado, ou automatizada, que é o processo de teste utilizando uma ferramenta para construir e produzir os testes em scripts que serão executados automaticamente.

Essas formas são bastante discutidas em suas vantagens e desvantagens. Os testes manuais têm um grande problema em relação ao tempo gasto para sua execução que é bastante superior ao tempo de execução dos testes automatizados, a automação de testes funcionais é indicada quando são muitos repetitivos, demorados e necessitam de um tempo e esforço considerável da equipe de teste. Ela possibilita uma redução no tempo e ciclo de testes, provocando um aumento indireto na cobertura do software, permitindo que os testadores foquem em outros tipos de testes que não podem ser automatizados, conseqüentemente, dando uma maior cobertura das funcionalidades do sistema.

Pressman (2006, p. 827), ressalta a importância de se ter uma ferramenta de automação de teste em um projeto de software na fase de desenvolvimento: "Teste de software freqüentemente é responsável por até 40% de todo o esforço despendido num projeto de desenvolvimento de software, as ferramentas que podem reduzir o tempo de teste (sem reduzir a eficácia) são muito valiosas".

2.2. Automação de Testes Funcionais

Segundo Fewster e Graham (1999), a automação de testes não é uma atividade que agregará maior qualidade aos testes, ele é apenas uma maneira mais rápida de se encontrar um defeito e tem a mesma eficiência se fosse executado manualmente.

Para Pettichord (2001), ela é uma tarefa que não agrega valor de imediato, pois tem um custo muito mais elevado para criar e manter os casos de testes em relação aos testes manuais, mas após ter todo o processo definido ele se torna mais rentável, já que o esforço utilizado para sua execução é uma pequena fração do que seria utilizado para executá-los manualmente.

Outro fator importante na visão de (INTHURN, 2001, p.51), é que mesmo a fase de teste sendo essencial no processo de desenvolvimento de software e de representar um custo para as empresas, em grande parte não proporciona os resultados esperados, uma vez que as rotinas de teste se tornam um trabalho maçante, repetitivo e que exige muito tempo e organização por parte dos testadores.

Este problema enfrentado por algumas empresas, e a busca direta pela qualidade oferecida em seu produto vêm criando uma consciência e necessidade de investimentos na automação dos testes, utilizando ferramentas próprias para esses fins. A Tabela 1 apresenta uma comparação entre o teste manual e o teste automatizado de 1.750 casos de testes com 700 defeitos existentes. Através da análise dos dados, Bartié (2002, p. 64) afirma que os testes automatizados são mais econômicos, rápidos e eficientes do que os testes manuais.

Etapas dos testes	Teste manual	Teste automatizado	Melhoria (%)
Planejamento	32	40	-25%
Definição de casos de testes	262	117	55%
Execução dos testes	466	23	95%
Conferencia dos testes	117	58	50%
Gerenciamento do erro	117	23	80%
Relatórios finais	96	16	83%
Duração total (em horas)	1.090	277	75%

Tabela 1 – Fonte Bartié (2002, p.64).

O sucesso da automação de testes não depende apenas da ferramenta a ser utilizada, mas principalmente do processo e abordagem de testes adotado, da habilidade do profissional

que irá automatizar os casos de testes, ou seja, quanto melhor for seu processo mais barato será sua implementação e manutenção.

2.2.1. Técnicas de automação de teste

Hoje na literatura já são apresentadas algumas técnicas para se automatizar os testes funcionais. Dentre elas temos: *record & playback*, programação de scripts, *data-driven* (técnica orientada a dados), *keyword-driven* (técnica orientada a palavra-chave) e decomposição funcional. Nesta seção serão comentadas essas técnicas com objetivo de oferecer um melhor entendimento para posteriormente saber quais serão aplicadas no estudo.

2.2.1.1. Record & Playback

É uma técnica que consiste em utilizar uma ferramenta de automação de teste, que capta as ações executadas pelo usuário sobre a interface gráfica de uma aplicação e converte essas ações em scripts em várias linguagens (dependendo da ferramenta) e podem ser executadas quantas vezes for desejado. Em cada execução a ferramenta repete os passos realizados pelo usuário exatamente igual à execução original. Cada caso de teste automatizado tem uma lógica igual ao caso de teste manual que inclui dados de entrada, resultados esperados, passo a passo para sua execução e as ações sobre a aplicação.

A principal vantagem dessa técnica é sua simplicidade e praticidade, utilizada para testes executados poucas vezes, no entanto quando pretende automatizar uma quantidade grande de conjuntos de casos de teste. Entretanto ela deixa a desejar em diversos aspectos, tais como: alto custo e dificuldade de manutenção, baixa taxa de reutilização, tempo de vida curto e alta sensibilidade a mudanças no software e no ambiente de teste, qualquer alteração na interface gráfica da aplicação poderia obrigar a regravação de todos os casos de testes existentes. (FEWSTER e GRAHAM, 1999), (FEWSTER, 2001), (HENDRICKSON, 1998), (MARICK, 1997) e (PETTICHORD, 2000)

2.2.1.2. Programação de Scripts

É uma extensão da técnica *record & playback*, onde os scripts são gravados e posteriormente editados para que desempenhem um comportamento diferente ao do script gravado original durante sua execução. É necessário que a ferramenta possibilite a edição dos scripts. Essa técnica permite que o testador copie o script de um caso de teste similar

modificando pontos isolados para a criação de um novo caso de teste sem a necessidade de uma nova gravação, permitindo uma maior taxa de reutilização, maior tempo de vida, maior manutenibilidade e maior robustez dos scripts de teste.

Apesar das vantagens sua aplicação isolada não traz tantos resultados, visto que para cada caso de teste deve ser programado um script contendo todo o procedimento do teste, dados e entrada e resultados esperados. (FEWSTER e GRAHAM, 1999), (HENDRICKSON, 1998), (IBM Rational) e (TERVO, 2001)

2.2.1.3. *Data-Driven* (Técnica Orientada a Dados)

Essa técnica consiste em extrair dos scripts de teste, os dados de entrada e saída dos testes, sendo armazenados em arquivos separados do script. Os scripts passam a conter apenas os procedimentos do teste (lógica de execução), as ações sobre a aplicação e procedimento de chamadas ao arquivo de dados quando necessário.

A principal vantagem dessa técnica é a facilidade de adicionar, modificar ou remover dados de teste, ou até mesmo casos de teste inteiros, com uma pequena manutenção dos scripts. Permite ainda que os projetistas e os automatizadores de teste trabalhem em diferentes níveis de abstração, visto que o projetista iria apenas elaborar os arquivos de com dados de teste, sem se preocupar com o nível de programação destes. (ZAMBELICH, 1998), (FEWSTER e GRAHAM, 1999), (HENDRICKSON, 1998), (IBM Rational) e (TERVO, 2001)

2.2.1.4. *Keyword-Driven* (Técnica Orientada a Palavras-Chaves)

Essa técnica consiste em extrair dos scripts de teste os procedimentos de teste que representam a lógica da execução. Estes passam a conter apenas as ações específicas que serão executadas sobre a aplicação e identificadas por palavras-chave. Tais ações assemelham-se com funções de um programa, podendo inclusive receber parâmetros, que são ativados pela palavra-chave a partir da execução de diferentes casos de teste. Os procedimentos dos testes são armazenados em um arquivo com um conjunto ordenado de palavras-chave e seus respectivos parâmetros.

A principal vantagem dessa técnica é a facilidade em adicionar, modificar ou remover passos de execução no procedimento de teste com pouca manutenção, já que os scripts não irão mais conter os procedimentos no próprio código e sim obter-los diretamente dos arquivos

de procedimentos de teste. Os projetistas de teste e o implementador podem trabalhar em diferentes níveis de abstração. (ZAMBELICH, 1998), (FEWSTER e GRAHAM, 1999), (HENDRICKSON, 1998), (IBM Rational) e (TERVO, 2001)

2.2.1.5. Decomposição Funcional

Zambelich (1998, p. 7), explica neste artigo como utilizar o método de decomposição funcional. Segundo ela o principal conceito atrás da “Decomposição Funcional”, visa à redução de todos os casos de testes, consistindo em escrever as funções relativas ao usuário, para executar tarefas fundamentais, os scripts com funções relativas às regras de negócio e “sub-rotinas” ou “utilitários”, scripts que executam tarefas independentes um do outro. Essas áreas fundamentais incluem:

- 1 Navegação
- 2 Funções Específicas do Negócio
- 3 Verificação de Dados
- 4 Navegação de Retorno

Para alcançar esse objetivo é necessário separar os dados das funções. Isto permite que o script automatizado seja escrito para executar apenas uma função do negócio, usando arquivos de dados tanto para fornecer as entradas como os resultados esperados (verificação).

O nível mais elevado é o *Driver Script*, ele é o motor do teste, contendo uma série de chamadas para um ou mais scripts de casos de teste que irão executar o teste completo automaticamente na ordem definida, inicializando a aplicação se necessário. Os scripts de casos de teste contêm a lógica do teste, que chama as funções mapeadas nos scripts das regras de negócio necessários para executar o aplicativo. Os scripts de sub-rotinas ou utilitários são chamados de acordo com a necessidade do driver, menu, e funções dos scripts das regras de negócios, e por ultimo funções gerais relativas ao usuário e acesso a telas que podem ser chamadas por qualquer um dos scripts acima. Abaixo podemos ver um exemplo na prática de como aplicar a técnica:

Os passos seguintes representam um caso de teste realizar pagamento:

1. Acesse a tela de pagamento através do menu principal
2. Faça o pagamento
3. Verifique se o pagamento atualizou o saldo da conta
4. Retorne ao menu principal
5. Acesse a tela de extrato através do menu principal
6. Verifique a atualização do saldo da conta
7. Acesse a tela do histórico das operações através da tela de resumo de contas
8. Verifique a atualização do histórico das operações
9. Retorne ao menu principal.

Um script de “regra de negócio” e um de “sub-rotinas”, podem ser escritos da seguinte maneira:

Pagamento:

- Inicie pelo menu principal
- Chame o método de navegação para acessar a tela de pagamento
- Leia o arquivo de dados contendo os dados específicos para este teste, insira esses dados
- Pressione o botão ou função-chave necessária para efetuar pagamento
- Ler arquivo de dados contendo os dados dos resultados esperados
- Compare os dados com os dados exibidos atualmente
- Escrever qualquer discrepância no relatório de erro
- Pressione o botão ou um link necessário para retornar ao menu principal, ou, se necessário, chame uma função de navegação para fazer isso.

Verificar Conta (Verificar Extrato e Histórico das transações):

- Inicie pelo menu principal
- Chame o método de navegação para acessar a tela de extrato de contas
- Leia o arquivo de dados contendo os dados de saída esperados
- Compare os dados com os dados mostrados na tela (saídas atuais)
- Escreva qualquer discrepância no relatório de erros
- Pressione o botão ou link para acessar o histórico das transações
- Ler o arquivo de dados contendo dados de saída esperados
- Compare esses dados com os dados mostrados na tela (saídas atuais)
- Escreva qualquer discrepância no relatório de erros
- Pressione o botão ou um link necessário para retornar ao menu principal, ou, se necessário, chame uma função de navegação para fazer isso.

Os Scripts de regra de negócios e sub-rotinas chamam as funções definidas pelo usuário para realizar a navegação. O script do caso de teste deve chamar esses dois se precisar realizar algum método referente à regra de negócio ou de utilidade geral. O Driver Script deve chamar o script do caso de teste o número de vezes que for necessário para executar todos os outros. Em cada caso, a única coisa que muda são os dados contidos nos arquivos de entrada que são lidos e processados pelos scripts de regra de negócio e os de sub-rotinas.

Usando este método, se necessários executar o processo com cinquenta tipos diferentes de pagamentos, a fim de verificar todas as condições possíveis, será necessário apenas quatro scripts que são reutilizáveis para todos os cinquenta casos:

1. O Driver Script
2. O script do caso de teste (realizar um pagamento e verificar os resultados)
3. O script das regras de negócio de "Pagamento"
4. O script de utilidade geral (verificar extrato da conta e histórico de transações).

Se fosse utilizado o método Record & Playback, iria ser construído cinquenta scripts diferentes, e dentro de cada um iria conter os dados de entrada e saída, que deveriam ser mantidos e dificultaria a manutenibilidade dos scripts.

Esse método, no entanto, exige que seja adicionado em arquivos os dados de entrada e saída necessários para cada teste e podem ser facilmente atualizados utilizando um bloco de nota ou editor de texto. Note que a atualização desses scripts não requer nenhum conhecimento na ferramenta, scripts, programação, o que significa que com essa técnica pode ser utilizada até mesmo por pessoas que não desempenha a função de testadores, enquanto o “técnico”, os testadores, podem criar e manter os scripts automatizados.

É importante notar que os scripts de sub-rotinas, os que verificam o extrato da conta e o histórico de transações, podem também ser utilizados por outros scripts de caso de teste e funções da regra de negócio, como por exemplo, um estorno de pagamento, isto significa que se precisarmos realizar cinquenta estornos de pagamento, precisaríamos desenvolver apenas três scripts adicionais.

1. O Driver Script
2. O script do caso de teste (estornar pagamento e verificar resultados)
3. O script da regra de negócio (estornar pagamento)

Considerando que já desenvolvemos quatro scripts anteriormente, é possível desenvolver rapidamente esse três novos a partir dos originais. É possível utilizar os scripts de utilidades gerais, “Verificar extrato de conta e histórico de transações” sem qualquer modificação.

Se for necessário utilizar diferentes contas, será preciso apenas atualizar o arquivo de dados e não o script todo, assim esse método se torna mais rentável do que o método Record & Playback.

2.4. Estratégias de sucesso na automação

Cem Kaner em (1997), publicou em seu artigo “Improving the Maintainability of Automated Test Suites”, algumas “boas práticas” no processo de automação de testes, discutidas por treze experientes testadores e destacaram os seguintes pontos:

1. Redefinir a gestão das expectativas em relação ao tempo dos benefícios da automação.
 - a. Há um grande benefício em automatizar um conjunto de testes de aceitação de build (também chamados “Smoke” testes), é possível executar esses cinquenta ou cem vezes durante o desenvolvimento de um release, mesmo se o tempo gasto for dez vezes mais para desenvolver que executar o cada teste a mão, e mais dez vezes do tempo gasto para manutenção, haverá um ganho equivalente entre trinta a oitenta execuções manuais.
 - b. É possível economizar tempo, reduzir o erro humano, e obter um monitoramento do que foi testado, nos casos em o sistema for testado com diversos dispositivos ou ambientes. Ex: Se for testada a compatibilidade do sistema com trinta impressoras, o custo da automação desses testes retorna em menos de uma semana.
 - c. Os testes de regressão facilitam o aferimento do desempenho do sistema em diferentes versões do mesmo.
2. Reconhecer que desenvolvimento de automação de teste é um desenvolvimento de software.
 - a. Não se pode desenvolver suítes de testes que irão ser utilizados em diversos release visando um baixo custo de manutenção, sem um planejamento claro e realístico.
 - b. A linguagem de programação deve ser simplificada, assim como todo o desenvolvimento do software.
 - c. Encarar cada caso de teste como um requisito (funcionalidade) do software.
 - d. Os programadores que neste caso serão os testadores devem: entender de requisitos, entender de programação, adotar uma arquitetura robusta para desenvolver, integrar e manter as funcionalidades, entre outras atribuições de qualquer desenvolvedor de software.

3. Utilizar uma arquitetura orientada a dados (*data-driven*)

- a. Essa técnica orientada a dados separa os dados de entrada dos scripts de teste em arquivos, quando os scripts são executados obtém os dados desses arquivos evitando que fiquem dentro do código dos scripts, isso aumenta consideravelmente a manutenibilidade dos scripts de teste.

4. Utilizar uma arquitetura baseada em um framework

É uma abordagem diferente, embora, muitas vezes utilize um conjunto de um ou mais estratégias de *data-driven*. Um *framework* isola a aplicação sob teste dos scripts de teste fornecendo um conjunto de funções em uma biblioteca compartilhada de funções. Os scripts de teste utilizam essas funções como se fossem comandos básicos da linguagem da ferramenta de automação, podendo assim programar os scripts independente da interface do usuário.

- a. Definir cada funcionalidade da aplicação
- b. Definir comandos ou funcionalidades da ferramenta da linguagem de programação.
- c. Definir conceitualmente e unificar tarefas que são feitas com frequência.
- d. Definir a complexidade das partes dos casos de testes que são utilizados em vários casos de teste.
- e. Definir utilitários e funções.

5. Reconhecer a realidade da equipe do projeto

- a. Muitos testadores não possuem experiência com programação eles são úteis para compor o esforço de teste da sua equipe, mas não para escrever códigos de automação.
- b. Não-programadores podem ser beneficiados utilizando uma abordagem dirigida a dado, eles poderão desenvolver processos de teste simplesmente introduzindo idéias em uma planilha.

6. Considerar a utilização de outros tipos de automação
 - a. A automação de testes de regressão através da interface pode trazer uma falsa ideia de cobertura que não existe, além de poder sobrecarregar a equipe deixando profissionais mais experientes criando e mantendo scripts ao invés de encontrar bugs.
 - b. As ferramentas de automação pela interface são úteis desde que se tenha um planejamento detalhado, profissionais capacitados e requer um investimento significativo.

2.5. Dificuldades na automação de testes

Cem Kaner, explica no mesmo artigo os principais problemas enfrentados ao automatizar os testes funcionais:

- **Automatizar não é uma atividade barata.**

Geralmente leva-se entre três a dez vezes mais tempo para criar, verificar e documentar minimamente os testes automatizados do que criá-los e executá-los uma vez o teste à mão, muitas vezes automatizar vale, mas para os testes que são executados apenas uma ou duas vezes, esta abordagem é ineficiente.

- **A automação de testes pode criar risco de custos adicionais.**

O custo de detectar e corrigir um erro aumenta ao passar do tempo. Se for utilizada a maior parte do tempo dos primeiros testes, escrevendo roteiros de teste, os erros significativos serão encontrados cada vez mais tarde, e o custo para corrigi-los já será elevado.

- **Testes automatizados podem não encontrar erros recorrentes.**

Os testes que são automatizados são testes que já passaram pelo sistema. Quantos novos bugs serão encontrados nesse caminho? Estima-se que variam entre seis a trinta por cento. Os números aumentam ao criar casos de testes, mas esses normalmente são testes manuais.

3. DESENVOLVIMENTO

O estudo foi aplicado em uma empresa de desenvolvimento de software para automação comercial e como seu processo de teste era pouco definido sem nenhum tipo automação, criamos um projeto para a concepção do processo de automação de teste dividido em cinco etapas: identificação dos casos de testes, projeto, construção, execução e verificação.

Para exemplificar melhor todo o projeto, e acompanhar o seu andamento, um diagrama das atividades realizadas em cada etapa é apresentado na Figura 1, com as informações de tempos e custos estimados para a realização das tarefas, a serem explicadas nas seções seguintes.

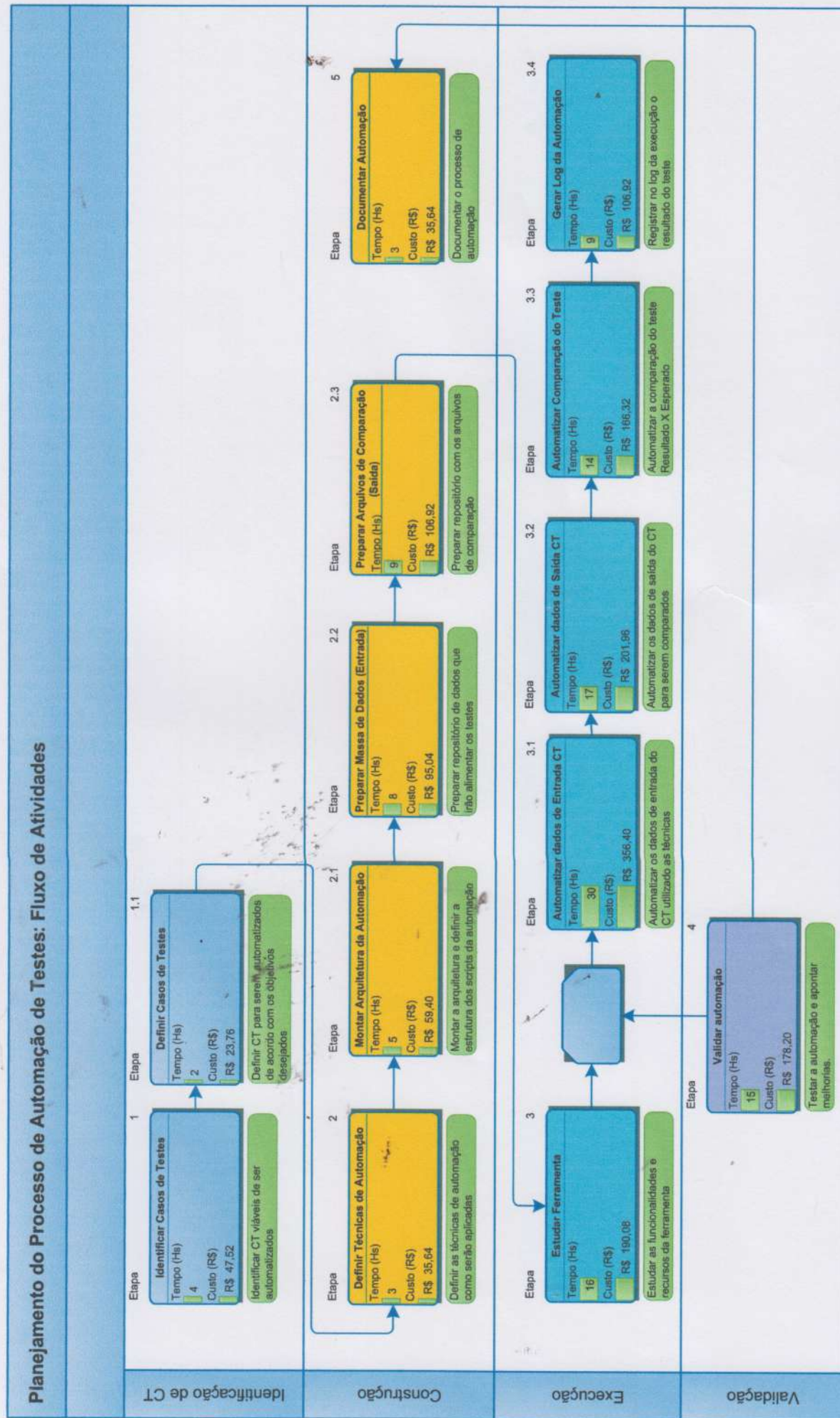


Figura 1 – Fluxo de atividades do projeto de automação – Fonte: Próprio Autor.

3.1. Identificação de Casos de Teste

Etapa 1: Identificar Casos de Teste

Identificação dos casos de teste possíveis e viáveis de ser automatizados. Os principais fatores considerados para a escolha foram o nível de complexidade da regra de negócio do caso de teste, tempo gasto na execução manual, número de repetições e o nível de incidência de erros, que foi analisado com o auxílio da ferramenta de gerencia de configuração da empresa.

Etapa 1.1: Definir Casos de Teste

Diante do levantamento feito na etapa anterior, verificou-se que as maiores incidências de erros ocorriam no módulo de vendas, principal módulo do sistema, mas devido à complexidade da regra de negócio não seria viável automatizar esses casos de teste de início, pois levaria muito tempo desenvolvendo até chegar aos resultados da automação, então foi adotada uma estratégia de automatizar os testes do módulo de cadastro, que a análise apontou como os que mais desperdiçavam tempo muitas vezes desnecessários. Assim a princípio, diminuiria o trabalho maçante e repetitivo dos testadores na execução dos casos de teste como alteração, exclusão e verificação de log, proporcionando maior liberdade e tempo para direcionar a força de trabalho nos testes dos outros módulos e relatórios do sistema, até que o processo de automação fosse totalmente validado. Desta maneira seria analisada a viabilidade de se automatizar também os outros casos de teste.

3.2. Construção

Etapa 2: Definir Técnicas de Automação.

Definição das técnicas utilizadas com base na teoria estudada e como essas serão aplicadas com base nos casos de testes.

Etapa 2.1: Montar Arquitetura da Automação

A arquitetura da automação é fundamentada em três elementos. O primeiro deles é a aplicação e o banco de dados modelo, que contém os dados válidos e não válidos que serão utilizados pelo segundo elemento, a ferramenta de automação de testes funcionais, na execução dos casos de teste. O último elemento é a interface gráfica da aplicação (GUI) e banco de dados em teste. Para um melhor entendimento do esquema montado observe a imagem na Figura 2:

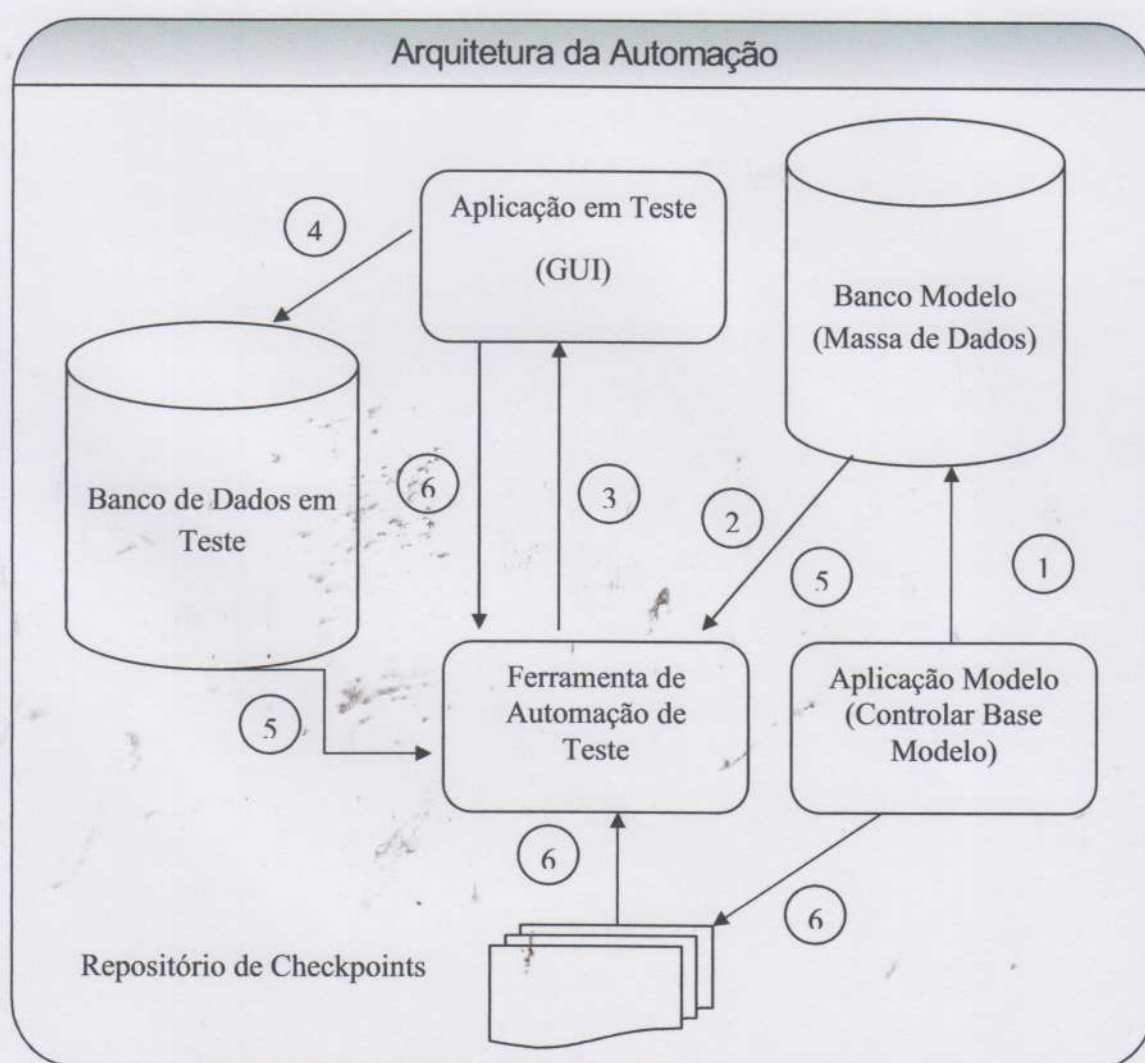


Figura 2 – Arquitetura do projeto de automação. – Fonte: Próprio Autor.

Nos parágrafos seguintes serão descritos os elementos do diagrama e sua função dentro da arquitetura definida para o projeto de automação de testes funcionais.

Os elementos da *Aplicação* e *Banco Modelo* correspondem à instância da aplicação e banco de dados em teste. Eles serão os modelos de comparação dos resultados obtidos na execução da automação, utilizados para inserir os dados de acordo com a regra de negócio de cada caso de teste. Para os dados inválidos o cadastro é feito diretamente na base de dados modelo, caso contrario a aplicação controladora não deixaria inserir o registro.

Por exemplo, para o caso de teste, "Cadastrar Cliente com CPF inválido", será inserido diretamente na base de dados modelo um cliente com CPF inválido. Um outro exemplo seria o caso de teste, "Cadastrar Cliente", onde será cadastrado um cliente utilizando a aplicação controladora.d

A *Ferramenta de Automação de Testes Funcionais* é o motor do teste, contendo os scripts com os algoritmos dos casos de teste, com funções de consultas dos dados do banco modelo, ações de navegação nas telas e inserir dados na aplicação. Além disso, é responsável pela comparação dos resultados da execução com os esperados e gerar um *log* da execução. Para tal é necessário que ela possua inteligência suficiente para realizar a mesma operação que seria realizada por um humano na execução manual.

A *Aplicação em Teste* e *Base de Dados em Teste* corresponde à aplicação que será testada com um banco de dados em teste inicialmente vazio, e será preenchido pela aplicação manipulada pela ferramenta de automação na execução dos casos de teste.

De acordo com os elementos apresentados, o seguinte fluxo de operações é realizado para a preparação e execução dos testes automáticos:

- 1 O testador cadastra as informações no banco de dados modelo através da aplicação controladora de acordo com o caso de teste que será executado. No caso do cadastro de dados inválidos, estes serão inseridos diretamente no banco modelo, já que a aplicação controladora tem que realizar a checagem do dado que será inserido. Nessas situações o testador captura uma tela que será utilizada posteriormente pela ferramenta de automação na comparação desses casos de testes com valores inválidos.

- 2 A ferramenta de automação obtém os dados do banco modelo através de consultas definidas na execução de cada caso de teste de acordo com a regra de negócio que será testada.

- 3 A ferramenta de automação manipula a aplicação em teste executando os casos de teste.

- 4 A aplicação em teste insere os dados da execução dos testes.
- 5 A ferramenta de automação compara os dados da base de dados em teste com o banco modelo após a execução dos testes.
- 6 A ferramenta de automação compara as imagens e arquivos gerados na execução do teste com as imagens e arquivos que foram gravadas pelo testador através da aplicação modelo e registra o resultado da comparação no log da execução.

Para melhor compreender a arquitetura do sistema que é composta por quatro *scripts* principais divididos em *Negócio*, que tem a sequência lógica do caso de teste e chama funções dos scripts de *Navegação*, para acessar a tela desejada e *Digitação*, para preencher os campos necessários na execução depois registrar no *LogExecução*. O script *Utilitários* tem as funções comuns a todos os scripts, como por exemplo, iniciar aplicação, executar selects no banco de dados, voltar ao menu principal, etc. O *DriverScript*, é responsável iniciar aplicação e guiar o teste chamando as funções do *Negócio*, de acordo com a Figura 3:

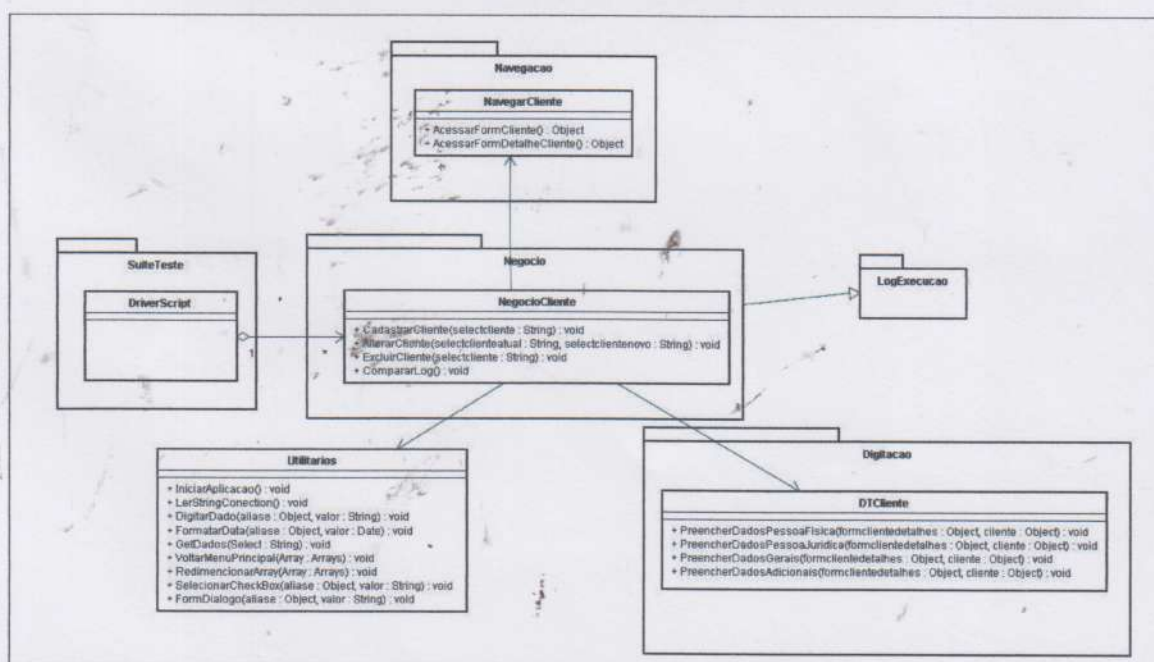


Figura 3 – Scripts utilizados pela arquitetura. – Fonte: Próprio Autor.

Neste exemplo, o *DriverScript* chama a função de negócio *CadastrarCliente* passando como parâmetro a consulta correspondente ao caso de teste que irá realizar. Neste ponto é

importante lembrar que a regra de negócio do caso de teste está na base de dados e aplicação modelo. Por exemplo, se o caso de teste for “CadastrarClienteComCPFInválido”, o DriverScript do caso de teste irá chamar a função cadastrar cliente passando a consulta correspondente ao cliente com CPF inválido.

A partir deste ponto a função CadastrarCliente, envia esta consulta (em formato SQL) para uma função de *Utilitários* que o executa no banco de dados modelo e retorna o cliente ou os clientes correspondentes. Em seguida continua a sequência de execução chamando as funções de acessar a tela e digitar os valores nos campos da tela correspondente comparando o resultado e gravando o resultado e gravando no log do teste até que não tenha mais objetos. Por fim, retorna para o menu principal e continua a execução do DriverScript.

Etapa 2.2: Preparar Massa de Dados (Entrada)

Esta etapa visa selecionar os dados de acordo com as técnicas de particionamento de equivalência, análise de valor limite e gráfico de causa e efeito explicadas previamente, que serão cadastrados na base modelo e depois utilizados na automação: eles serão a base para a regra de negócio do caso de teste.

A escolha de deixar a regra de negócio do teste na aplicação e na base de dados modelo deve-se à complexidade do sistema, mas para que o resultado seja confiável é necessário que a operação seja validada pelo testador.

Por exemplo, qualquer operação na aplicação modelo gera um *log* na base de dados modelo para que esse seja usado como comparação com o *log* gerado pela aplicação em teste na base de dados teste na execução da automação.

Etapa 2.3: Preparar Arquivos para Comparação (Saída)

O testador executa o caso de teste na aplicação modelo e grava o resultado esperado e põem em um repositório depois de ser validado com a identificação do caso de teste executado manualmente, essa gravação pode ser feita com uma foto da tela, ou arquivos de texto simples ou XML gerados pela aplicação. Por exemplo, ao cadastrar um cliente com o CPF inválido a aplicação deve exibir uma imagem com a mensagem “CPF inválido” a captura da tela é gravada e colocada no repositório. Então a ferramenta de automação executa o

mesmo caso de teste e captura uma nova tela na execução e compara com o arquivo do repositório e valida ou não o teste.

Essa vantagem também será percebida na automação dos casos de teste do módulo de vendas, assim fica fácil checar os cálculos efetuados, a venda primeiramente será realizada na aplicação modelo gerando um arquivo de texto com os dados da nota fiscal que será colocado em um repositório com a indicação do caso de teste equivalente a essa nota fiscal. A ferramenta automação irá simular a mesma venda na aplicação em teste depois comparar os arquivos e saber o teste foi aprovado.

3.3. Execução

Para aplicação do estudo foi utilizada a ferramenta **TestComplete** distribuída pela empresa AutomatedQA (www.automatedqa.com), focada em produtos que auxiliam empresas a acelerar o desenvolvimento e os processos de teste e entregar soluções de alta qualidade em menor tempo aos seus cliente, reduzindo drasticamente os custos com correções. Dentre as suas principais ferramentas podemos destacar:

TestComplete: atualmente na versão 7, é uma ferramenta que permite criar, executar e gerenciar testes automatizados, em softwares desenvolvidas em Win-32 ou Web, é indicada para equipes de qualidade que precisam realizar muitos teste, ter um acompanhamento rápido e entregar o software em qualquer momento.

AQTime: atualmente na versão 6, é uma ferramenta com o objetivo fundamental de ajudar a compreender como os software estão sendo executados, coletando informações de forma resumida ou detalhada da alocação de recursos em tempo de execução, podendo indicar a linha exata do código fonte que apresenta algum gargalo de desempenho ou vazamento de memória, se integra as IDE *Microsoft Visual Studio*, *Borland Developer Studio* ou *CodeGear RAD Studio*.

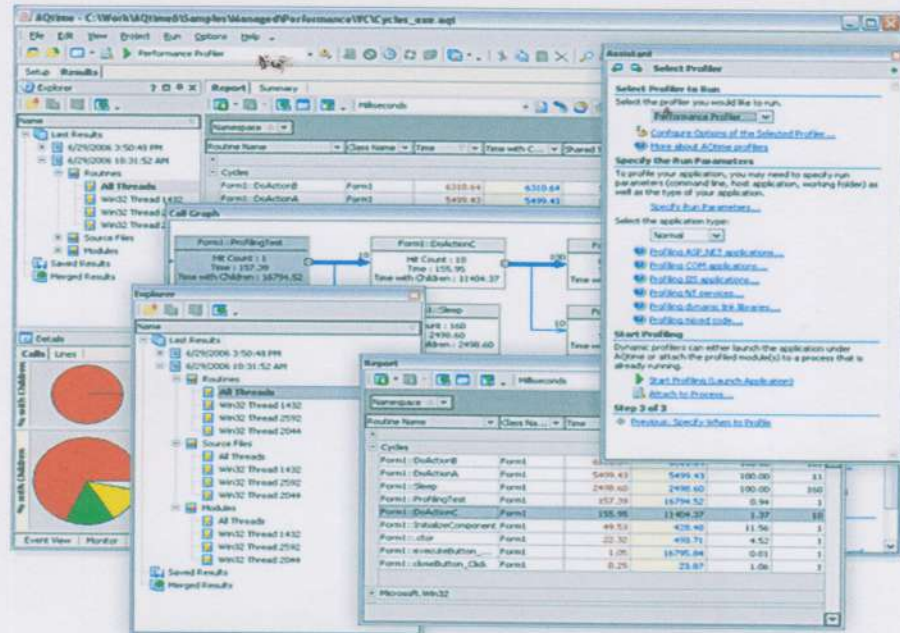


Figura 4 – AutomatedQA QATime – Fonte: www.automatedqa.com

TestComplete, é um ambiente de desenvolvimento de testes automatizados para uma ampla gama de tipos de aplicações e tecnologias, incluindo do Windows. NET, WPF, Visual C, Visual Basic, Delphi, C Builder, Java e aplicações web, servidores web e web services, podendo gerar scripts de testes nas linguagens: C#, C++, Java, Delphi e Visual Basic.

Suporta vários tipos de testes, como testes de unidade, regressão funcional, testes de GUI, testes de carga, stress e testes de escalabilidade, que podem ser executados por usuários virtuais simulando várias estações de trabalho na mesma aplicação simultaneamente.

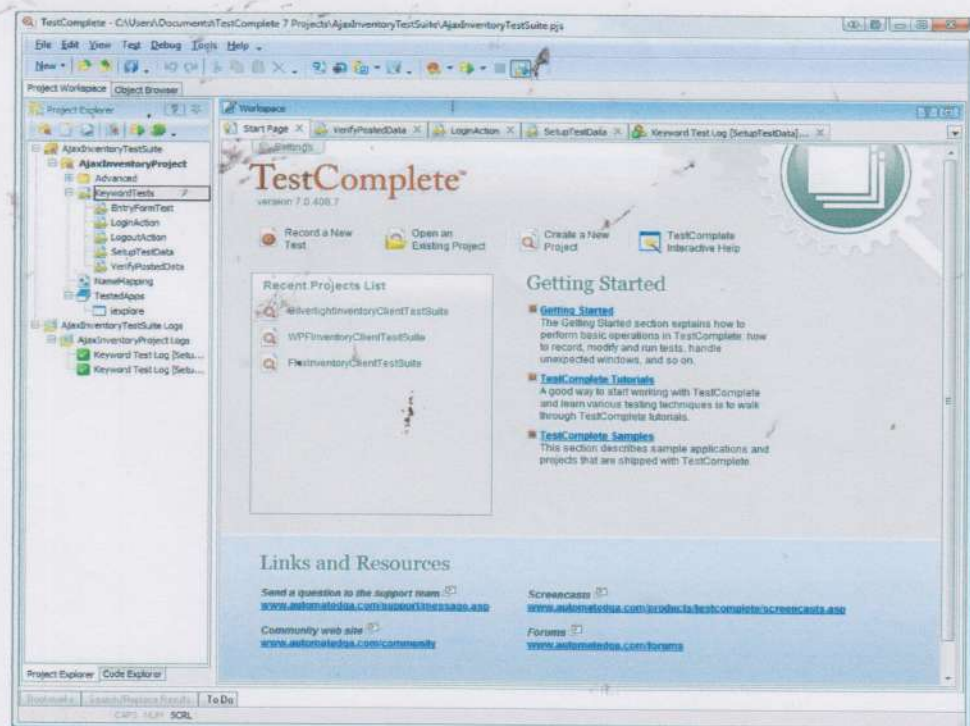


Figura 5 – AutomatedQA TestComplete.- Fonte: www.automatedqa.com

Etapa 3: Estudar Ferramenta de Automação

Nesta etapa se fez necessário estudar a ferramenta, entender o seu funcionamento e funcionalidades para então dar início ao processo de automação. Ela pode ser descartada se os automatizadores possuírem conhecimentos suficientes para realizar a automação.

Nos experimentos realizados com a equipe de desenvolvimento da empresa, a ferramenta se mostrou bastante fácil e intuitiva e em apenas 16 horas de estudo foi possível obter o conhecimento básico para dar início a programação da automação.

O projeto foi desenvolvido utilizando a linguagem Java para geração do código. Poucas funcionalidades da ferramenta foram utilizadas para não deixar o processo de automação muito dependente. Sendo assim, foram utilizadas apenas as principais:

NameMapping: Utilizada para mapear a tela, com essa função é possível manipular formulário, edits, botões. Etc.

Script: Programar as funções da automação.

TestedApps: Iniciar as aplicações utilizadas nos testes.

Stores: Repositório de modelos utilizados para comparação na execução dos testes, esses modelos podem ser tabelas, arquivos txt ou xml, objetos, etc.

Logs: Registrar os logs das execuções dos testes.

CheckPoints: Criar pontos de comparação no teste.

Etapa 3.1: Automatizar entrada do caso de teste:

Cadastrar no banco modelo os dados válidos e não válidos, para serem utilizados na execução da automação.

Foi criado Views no banco modelo para que nas consultas realizadas pela ferramenta de automação retornassem todos os dados necessários para o caso de teste. A função de digitação associa o valor de retorno a um determinado objeto na tela e digita o valor recebido.

Etapa 3.2: Automatizar saída do caso de teste:

Inserir nos *Stores* os dados que serão utilizados pelos pontos de verificação na comparação dos resultados da execução dos casos de teste.

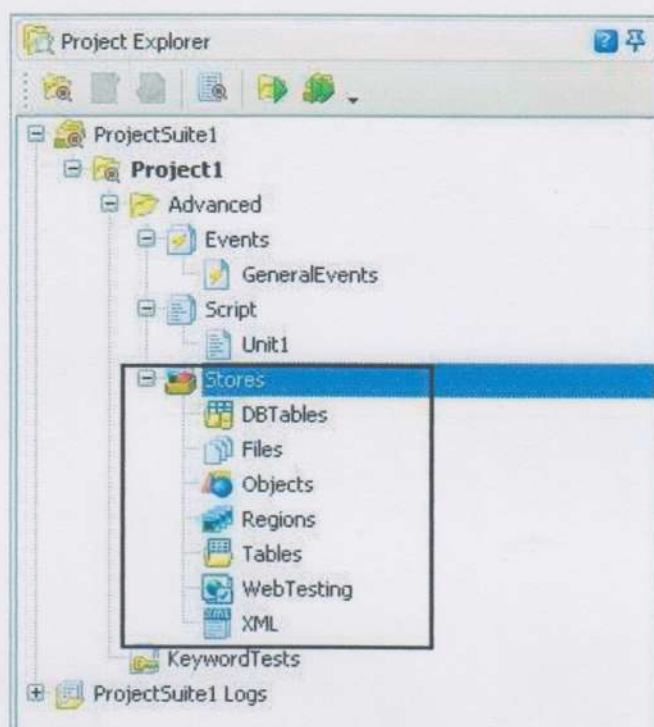


Figura 6 – Árvore dos Stores – Fonte: Próprio Autor

Etapa 3.3: Automatizar os Checkpoints (pontos de verificação):

Criar os pontos de verificação do teste, para comparar os dados que estão no repositório com os resultados na execução no teste utilizando os Checkpoints.

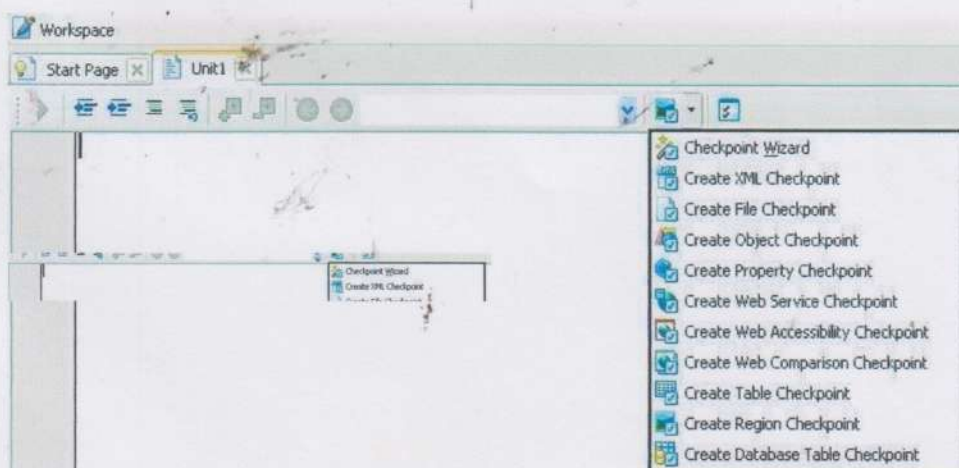


Figura 7 – Lista de Checkpoint. – Fonte: Próprio Autor

Etapa 3.4: Gerar Log da automação:

Gerar o Log da automação com mensagens de sucesso ou erro e imagens para facilitar na análise da execução, para isso a ferramenta oferece os Log.Erro, Log.Message, Log.Picture, etc.

3.1. Validação**Etapa 4: Validar automação do caso de teste:**

Essa etapa consumiu 11,11% do tempo da automação, para executar os casos de teste e verificar se o resultado com a automação foi o esperado.

4. RESULTADOS OBTIDOS

O experimento foi realizado apenas em uma pequena porcentagem dos casos de teste de regressão selecionados em uma suíte de teste, que são re-executados a cada liberação de versão para garantir que o que foi desenvolvido não afetou as outras funcionalidades já existentes no sistema.

O principal objetivo é comparar a efetividade da automação analisando tempo e custo comparando com o processo de execução manual utilizado no passado. Neste capítulo serão apresentados os resultados encontrados e suas avaliações.

4.1. Contexto

A fábrica de software tem soluções na área de automação comercial, seu principal produto P1 possui uma carteira de aproximadamente 400 clientes gerando uma grande demanda de requisitos expressos pelos clientes ou atender as constantes mudanças solicitadas pelo governo.

Atualmente a empresa não dispõe de nenhum processo de execução automatizada, para cada release liberado são re-executados 490 casos de teste representando um esforço de 176 Horas por dois testadores, tarefa maçante que consome cerca de 70% da força de trabalho da equipe de teste da organização composta por três testadores que ver a necessidade de expandir e aumentar a complexidade dos testes do produto P1, diminuir o tempo na identificação dos erros e ainda utilizar essa equipe em outros projetos e aplicar os tipos de teste e não apenas o funcional.

4.1.1 Escopo do Processo de Automação

Para o estudo foi selecionados os casos de teste do módulo de cadastro que correspondem a 15% do ciclo do teste de regressão, com objetivo de liberar os testadores da tarefa monótona de aproximadamente 4 horas e iniciar uma cultura de automação de teste em seus projetos.

4.2. Análise dos resultados

Os cálculos para a estimativa para o retorno do investimento da automação foram baseados no artigo de VARGHESE, que explica que devem ser comparados:

- Esforço considerado para manutenção corretiva e evolutiva. Foi definido 2% a mais do tempo gasto na automação por ser o projeto pioneiro e está em constante correção e evolução.
- Custo homem/hora de um programador dedicado a automação dos testes equivalente R\$ 2.092,00 por mês ou R\$ 11,88 por hora, sem incluir os encargos.
- Hardware utilizado na automação e na execução da automação, um computador com custo de mercado de dois mil e trezentos reais.

Foram levantados todos os custos da automação para fazer o cálculo do tempo de retorno do investimento da fábrica de software e serão apresentados a seguir.

4.2.1 Custo da Automação

Seguindo a orientação do artigo os custos foram divididos em fixos e variáveis. Os custos fixos correspondem ao investimento empregado uma única vez em todo o projeto que foi dividido em etapas para ajudar nos cálculos conforme apresentado no fluxo de atividades pag. 25 e na tabela 2 abaixo:

Etapa	Nome	Descrição	Tempo (Hs)	Custo (R\$)
1	Identificar Casos de Testes	Identificar CT viáveis de ser automatizados	4	R\$ 47,52
1.1	Definir Casos de Testes	Definir CT para serem automatizados de acordo com os objetivos desejados	2	R\$ 23,76
2	Definir Técnicas de Automação	Definir as técnicas de automação como serão aplicadas	3	R\$ 35,64
2.1	Montar Arquitetura da Automação	Montar a arquitetura e definir a estrutura dos scripts da automação	5	R\$ 59,40
2.2	Preparar Massa de Dados (Entrada)	Preparar repositório de dados que irão alimentar os testes	8	R\$ 95,04
2.3	Preparar Arquivos de Comparação (Saída)	Preparar repositório com os arquivos de comparação	9	R\$ 106,92
3	Estudar Ferramenta	Estudar as funcionalidades e recursos da ferramenta	16	R\$ 190,08
3.1	Automatizar dados de Entrada CT	Automatizar os dados de entrada do CT utilizando as técnicas	30	R\$ 356,40
3.2	Automatizar dados de Saída CT	Automatizar os dados de saída do CT para serem comparados	17	R\$ 201,96

3.3	Automatizar Comparação do Teste	Automatizar a comparação do teste Resultado X Esperado	14	R\$ 166,32
3.4	Gerar Log da Automação	Registrar no log da execução o resultado do teste	9	R\$ 106,92
4	Validar automação	Testar a automação e apontar melhorias.	15	R\$ 178,20
5	Documentar Automação	Documentar o processo de automação	3	R\$ 35,64
Total			135	R\$ 1.603,80

Tabela 2 – Custo das atividades.

OBS: Os dados apresentados correspondem especificamente a esse estudo de caso, o que significa que pode haver variações de tempo e custo dependendo do objetivo final da automação.

Com os valores corresponde a cada atividade somamos com os custos de investimentos, com ferramenta e hardware, veja a tabela 3 abaixo:

- Custo da Ferramenta: uma licença da AutomatedQA TestComplete Enterprise no valor de \$ 1.999,00. Foi utilizada a taxa de câmbio R\$ 1,83 para o cálculo do valor em R\$.
- Custo com planejamento: utilização de um automatizador para o planejamento da automação nas atividades da etapa 1.
- Custo com construção da automação: utilização de um automatizador para construir a arquitetura da automação e estrutura dos scripts, nas atividades da etapa 2.
- Custo com execução: utilização de um automatizador na programação dos scripts das atividades da etapa 3.
- Custo com validação: esforço para testar e validar os scripts de teste na atividade da etapa 4.

- Custo com documentação: esforço para documentar os scripts de teste na atividade da etapa 5.
- Custo com infra-estrutura: um computador para ser utilizado na execução da automação dos testes.

Custos Fixos	Valor
Custo da Ferramenta*	R\$ 3.658,17
Planejamento	R\$ 71,28
Construção da Automação	R\$ 297,00
Execução	R\$ 1.021,68
Validação	R\$ 178,20
Documentação	R\$ 35,64
Infra-Estrutura	R\$ 2.300,00
Total	R\$ 7.561,97

Tabela 3 – Custos fixos da automação.

Os custos variáveis foram definidos baseados na tabela 4:

- Custo com a manutenção dos scripts: para esse projeto foi definido 2% do valor investido na construção dos scripts.
- Custo da análise da execução: custo com um automatizador durante 15 minutos para coletar do log da execução do teste automatizado os casos de teste que apresentaram erros.

Custos Variáveis	Valor
Manutenção dos Scripts	R\$ 5,94
Análise da Execução	R\$ 2,97
Total	R\$ 8,91

Tabela 4 – Custos variáveis da automação.

4.2.2 Custo dos Testes Manuais

Na execução manual os custos são compostos com a construção dos planos dos casos de teste e a execução manual. Para o estudo não foi possível estimar o custo com a construção

do plano de teste, pois foram utilizados os que já haviam na fábrica de software com isso foram considerados apenas os custos com a execução.

- **Custo da Execução:** pelos relatos dos testadores da fábrica de software o tempo médio da execução dos testes no módulo de cadastro era de 3 minutos, multiplicado pela quantidade de casos de teste, 70, representava um total de 210 minutos ou 3,5 Horas com um valor de R\$ 8,00 por hora de um testador com salário de R\$ 1.408,00.

A tabela 5 abaixo mostra o número de campos preenchidos na execução dos testes de cadastro.

Cadastro	Nº de Campos
Empresa	30
Produto	68
Cliente	76
Funcionário	54
Fornecedores	38
Fabricantes	23
Tabelas	115
Total	404

Tabela 5 – Nº de campos de cadastros.

Para prosseguir com os cálculos explicados por VARGHESE em seu artigo, é necessário calcular a melhoria no custo da execução dos testes automatizados em comparação com os testes manuais, os custos com a construção dos scripts devem ser diluídos a cada novo ciclo dos testes executado.

A melhoria significativa no processo de automação é notada na redução do tempo de execução dos testes, outros ganhos observados, mas não inseridos nos cálculos, foi à confiança dos testes serem executados por uma máquina em qualquer momento desejado sem interromper o trabalho dos testadores se tornando assim um testador auxiliar somando a força de trabalho da empresa.

4.2.3 Melhoria no Tempo de Execução dos Testes

Como já foi comentado, o tempo total da execução dos testes de forma manual é de 3,5 Horas ou 12.600 segundos.

Fazendo um comparativo com a execução da suíte com os 70 casos de testes de forma automatizada que levaram em média 32,66 minutos ou 1.959,60 segundos chegamos a uma melhoria de 84,45% no tempo de execução.

4.2.4 Melhoria no Custo de Execução dos Testes

É possível chegar ao valor correspondente a melhoria dos custos de execução dos testes apenas diminuindo o custo da execução manual pelo custo da execução automatizada e dividir pelo custo da execução automatizada.

Sabendo que o custo da execução manual equivale à R\$ 28,00 e o custo da execução automatizada à R\$ 18,32, observamos uma melhoria de 68,18% nos custos com a execução.

Redução de custo da Execução (A - B / A)		68,18%
A	Execução Manual	R\$ 28,00
B	Execução Automatica	R\$ 8,91
(A - B)		R\$ 19,09

Tabela 6 – Redução de custo da execução.

4.2.5 Retorno do Investimento

Para VARGHESE o tempo que a empresa levaria para ter o retorno investimento é obtido pela conforme a tabela 7 abaixo:

Nº de Ciclos para o retorno do Investimento (A / B * C)		396,12
A	Custo Fixo da Automação	R\$ 7.561,97
B	C. dos Teste Manuais	R\$ 28,00
C	% de redução de custo	68,18%

Tabela 7 – Retorno do investimento.

Sendo assim, seria necessária a execução de 395,12 ciclos de forma automatizada, o equivalente à aproximadamente dezesseis anos e meio, que pode ser considerado absurdo se forem executados duas vezes a cada mês como é executado manualmente, mas tendo a

possibilidade de se executar todos os dias do ano esse número cairia para aproximadamente um ano e um mês.

Os números devem-se ao fato de que para um investimento inicial alto com a ferramenta e a infra-estrutura da automação e a discrepante quantidade de casos de teste utilizados no estudo, podendo se tornar inviável o investimento se não for dada uma continuidade e a criação de novos casos de testes, para empresa este investimento retornará gradativamente com automatização de outros projetos além de atingir seu objetivo de iniciar o processo de automação e poder utilizar sua força de trabalho para aplicar outros tipos de teste não apenas o funcional.

5. CONCLUSÕES E PERSPECTIVAS

Teste de software vem se tornando cada vez mais importante no cenário de desenvolvimento atual, sendo encarado pelas *Software House* como um diferencial na apresentação e propaganda dos seus produtos, proporcionando uma garantia ao cliente ou em outros casos para se adequar às exigências dos modelos de referência em processo de desenvolvimento de software como CMMI e MPS-BR.

Essa verificação do produto de forma manual, pode se tornar inviável dependendo do nível de complexidade do sistema, sendo necessária uma equipe de teste enorme para re-testar todo o sistema e garantir a qualidade de versão a cada nova solicitação desenvolvida e que pode complicar, se o desenvolvimento for aos moldes das metodologias ágeis, com isso um processo de teste automatizado pode ajudar e garantir essa verificação ininterruptamente, com acompanhamento rápido e entrega de software em qualquer momento.

Este Trabalho possibilitou à fábrica de software o início de uma cultura de automação de testes, mesmo em uma equipe sem nenhuma experiência no processo de automatização, fato este atualmente, comum entre as empresas de desenvolvimento de software. O experimento vem para auxiliar equipes de teste com este perfil que pretende ter um melhor aproveitamento da sua força de trabalho.

A construção da arquitetura e aplicação das técnicas se mostrou eficientes, com o reuso do código e maior independência de execução, que possibilitou a execução dos testes ininterruptamente sem a presença de um observador.

A ferramenta *TestComplete* permitiu uma aprendizagem fácil e intuitiva com muitos recursos e um grande potencial para a finalidade que se propõe, podendo também automatizar outros tipos de teste e não apenas o funcional.

5.1. Dificuldades Encontradas

Entre as dificuldades encontradas é importante destacar a falta conhecimento prévio na ferramenta de automação *TestComplete*, que dificultou um pouco na criação dos scripts para obter o máximo de benefício, bem como, o processo de teste pouco definido utilizado na

fábrica de software, que atrapalhou em alguns momentos a análise para identificação dos casos de testes para serem automatizados.

5.2. Trabalhos Futuros

Para dar continuidade ao trabalho podemos identificar alguns pontos, como, a possibilidade de adaptar a automação com o controle de versão do produto para iniciar o processo de teste sem a intervenção humana, aplicando teste na integração do código escrito pelos desenvolvedores.

Criar formas de integrar os métodos de desenvolvimento ágil, como por exemplo, *Scrum* e *Extreme Programming* com a automação de teste, para que a agilidade na verificação do produto acompanhe o seu desenvolvimento, tendo em vista a constante aderência das *Software House* a essas metodologias.

6. BIBLIOGRAFIA

AutomatedQA TestComplete, Disponível em:

<http://www.automatedqa.com/products/testcomplete/>. Acessado em: 24/10/2009.

BARTIÉ, Alexandre. Garantia da qualidade de software. Campus, 2002.

Borland Segue SilkTest, Disponível em:

<http://www.borland.com/us/products/silk/silktest/index.html>. Acessado em 24/10/2009.

DANG, David. Test Automation: The Promise Versus the Reality. Disponível em:

<http://www.stickyminds.com/getfile.asp?ot=XML&id=10429&fn=XUS10356519file1%2Epdf>. Acessado em: 09/05/2009.

FEWSTER, M., GRAHAM D., Software Test Automation, Addison-Wesley, 1999.

FEWSTER, M., "Common Mistakes in Test Automation", Proceedings of Fall Test

GLENFORD, J. Myers, "The Arte of Software Testing", 2004.

HENDRICKSON, E., "The Differences Between Test Automation Success And Failure", Proceedings of STAR West, 1998.

IEEE Std 829-1998 Standard for Software Test Documentation.

IBM Rational. Disponível em: <http://www-306.ibm.com/software/rational/>.

IBM Rational Functional Tester. Disponível em: <http://www-01.ibm.com/software/awdtools/tester/functional/index.html>.

Acessado em: 24/10/2009.

INTHURN, Cândida. Qualidade & teste de software. Florianópolis: VisualBooks, 2001.

KANER, C.; Falk, J.; Nguyen, HQ. Testing Computer Software. Second Edition, Willey, 1999.

KELLY, Michael. The ROI of Test Automation. Disponível em:

<http://www.stickyminds.com/getfile.asp?ot=XML&id=8502&fn=XDD8502filelistfilename1%2Epdf>. Acessado em: 09/05/2009. Automation Conference, 2001.

KELLY, Michael. Framework automation with IBM Rational Functional Tester: Data-driven. Disponível em: http://www-128.ibm.com/developerworks/rational/library/05/1108_kelly. Acessado em: 09/05/2009.

KANER, Cem. Improving the Maintainability of Automated Test Suites, 1997. Disponível em: <http://www.kaner.com/lawst1.htm>. Acessado em: 09/05/2009.

KANER, Cem. Architectures of Test Automation, 2000. Disponível em: <http://www.kaner.com/testarch.html>. Acessado em: 09/05/2009.

MARICK, B., "Classic Testing Mistakes", Proc. of STAR Conference, 1997.

MOLINARI, Leonardo. Teste de Software: Produzindo Sistemas Melhores e Mais Confiáveis. Érica, 2003.

PETTICHORD, B., "*Capture Replay - A Foolish Test Strategy*", Proc. of STAR West, 2000.

PETTICHORD, Bret. Success with Test Automation, 2001. Disponível em: <http://www.io.com/%7Ewazmo/succpap.htm>. Acessado em: 09/05/2009.
Disponível em: <http://www.kaner.com/lawst1.htm>. Acessado em: 09/05/2009.

PEZZÈ, M. & YOUNG, M., "Teste e Análise de Software" – Bookman, 2008.

PRESSMAN, Roger S. Engenharia de Software, Sexta Edição, McGraw-Hill, 2006.

ROCHA, A. R. C., MALDONADO, J. C. & Weber, K. C., "*Qualidade de software – Teoria e prática*", Prentice Hall, 2001.

SOMMERVILLE, Ian. Engenharia de Software, 6ª Edição, Addison Wesley, 2003.

TERVO, B., "*Standards For Test Automation*", Proc. of STAR East, 2001.

VARGHESE, Jose. Test Automation – An ROI based Approach. Disponível em: http://www.qavision.com/expertzone/the_roi_of_test_automation.pdf. Acessado em: 01/11/2009.

ZAMBELICH, Keith. Totally Data-Driven Automated Testing, 1998. Disponível em: www.automation.org.uk/downloads/documentation/white_papers-totally_data_driven_automated_testing.doc. Acessado em: 09/05/2009.