REVISTA DE CIÊNCIAS EXATAS E TECNOLOGIA

AUTOMAÇÃO DE TESTES FUNCIONAIS
Testes funcionais automatizados de software

Paulo César Barreto da Silva, Thiago Salhab Alves, Elisângela Andrade Bruno – *Faculdade Anhanguera de Santa Barbara*

RESUMO: Os Testes Funcionais permitem que os testes ocorram de uma forma mais eficiente e rápida, possibilitando encontrar as inconformidades do software em relação aos requisitos do sistema. Este texto apresenta os principais conceitos dos Testes Funcionais de Software e alguns detalhes extraídos de sua utilização na monografia do trabalho de conclusão de curso para obtenção do titulo de Bacharel e Ciência da Computação sobre Automatização de Testes Funcionais, utilizando a ferramenta de automatização RATIONAL ROBOT, sendo que seus scripts, executados em uma aplicação Teste, utilizam o Framework CDK (CPqD Development Kit) do CPqD (Centro de Pesquisa e Desenvolvimento).

PALAVRAS-CHAVE:
Automatização, CDK, Testes Funcionais, Rational Robot e scripts.

ABSTRACT: The Functional Tests allow testing to occur in a more efficient and quick, allowing the software to find non-conformities with the requirements of the system. This paper presents the main concepts of the Functional Testing of Software and some details taken from his monograph on the use of completion of course work to obtain the title of Bachelor of Computer Science and Automation Functional Tests using the automation tool RATIONAL ROBOT, is that your scripts run in a test application, use the Framework CDK (CPqD Development Kit) CPqD (Centre of Research and Development).

KEYWORDS:
Automation, CDK, Functional Testing, Rational Robot and scripts.

Artigo Original
Recebido em: 19/03/2011
Avaliado em: 14/02/2014
Publicado em: 28/04/2014

Publicação
Anhanguera Educacional Ltda.

Coordenação
Instituto de Pesquisas Aplicadas e Desenvolvimento Educacional - IPADE

Correspondência
Sistema Anhanguera de Revistas Eletrônicas - SARE
rc.ipade@anhanguera.com

1. TESTES FUNCIONAIS

Atualmente os softwares são empregados em todos os seguimentos da sociedade, tanto para sistemas cujos requisitos sejam simples quanto para aplicações sofisticadas e de grande complexidade. Um exemplo de sistema complexo são os softwares de uso exclusivo para a área de medicina.

Com a ampla utilização das tecnologias da informação, a qualidade do produto tem se tornado imprescindível nos processos de desenvolvimento de aplicações e no momento de avaliação do projeto. O teste de software, cujo objetivo é revelar a presença de defeitos e aumentar a confiança sobre o software, é considerado um elemento crítico para a garantia da qualidade do produto. (VILELA a, 2005)(PRESSMAN, 1995).

As atividades de testes podem muitas vezes se tornar exaustivas e trabalhosas, dificultando assim a execução dos testes de forma adequada para a análise de qualidade. Com o objetivo de melhorar a qualidade da análise e o tempo de execução dos testes, foram criados os testes automatizados, que proporcionam a execução dos testes mais rapidamente, e com maior cobertura do software (SOMMERVILLE, 2007).

2. CONCEITO DE TESTES DE SOFTWARE

De acordo com Pressman, autor do livro Engenharia de Software, o principal objetivo dos testes de software é a localização de erros, falhas, defeitos e a verificação das funcionalidades do software em desenvolvimento ou finalizado.

Erro é um estado de execução que impede o software de continuar, pode ser causado por algo externo ao software ou falhas na grafia do código fontes, tais como ausência de ponto-e-virgula e parâmetros.

Uma Falha é um evento notável onde o sistema viola suas especificações, é a existência aparente de um defeito. (VILELA a, 2005) (RICARTE, 2006).

Defeito é uma deficiência mecânica ou algorítmica (deficiência no código) que se ativada pode levar a uma falha.

Através do processo de testes, busca-se avaliar se estas funcionalidades estão aparentemente trabalhando de acordo com as especificações e requisitos do projeto, garantindo que o software atinja o nível de qualidade esperada pelos interessados no produto (PRESSMAN, 1995).

3. PRINCIPIOS DOS TESTES

Segundo Brunelli, os componentes essenciais para o teste de software de um programa podem ser classificados em 5 itens:

- Executável do programa (código do programa compilado);
- Relação dos comportamentos esperados;
- Definição das abordagens de visualização dos comportamentos;
- Descrição das funções;
- Definição de observação se o comportamento executou da forma esperada.

O levantamento dos comportamentos esperados é o que proporciona maior dificuldade na elaboração dos testes, que deve ser feito de forma cautelosa e de acordo com os requisitos do sistema. Portanto a definição de um projeto de casos de testes é extremamente importante (PRESSMAN, 1995), pois um caso de teste bem elaborado é aquele que encontra um erro que ainda não tenha sido encontrado anteriormente (SOMMERVILLE, 2007).

Enfim os testes de software visam mostrar aos clientes que o software está funcional de acordo com as especificações propostas dentro dos limites de confiabilidade. Sendo que os testes têm como seu principal objetivo mostrar os erros e não deixar de demonstrá-los (PFLEEGER, 2004).

4. O PROCESSO DE TESTES

Há um momento em que faz necessária a representação das atividades, objetos, papeis e padrões na organização do processo de desenvolvimento.

O Processo de Teste, como qualquer outro processo, necessita ser refinado continuamente, permitindo seu amadurecimento, de maneira a ampliar sua atuação e permitir aos envolvidos no projeto uma maior visibilidade e organização dos seus trabalhos. O que se deseja nesta etapa é uma maior agilidade e controle operacional dos projetos de testes.

Realizar testes de software é uma tarefa bastante complexa quando se desconhece o que é qualidade. Nas industrias automobilísticas, como é o caso da maioria das grandes industrias, qualidade está intimamente associada a custo de retrabalho.

No inicio da década de 60, por volta do ano de 1962, foi criado no Brasil um comitê especifico para trabalhar a padronização das normas e regras que padronizam sua implantação nas empresas. A qualidade, muitas vezes associada a certificação como ISO 9001, CMM-I e tantas outras que existem por ai, não passam de formalizações de boas práticas que com o passar do tempo foram aperfeiçoadas e implementadas de forma comum.

O processo de Testes de Software deve contemplar, além de um roteiro com objetivos bastante claros, a declaração dos itens a serem avaliados e quais são os índices esperados, por exemplo, defeitos por número de funções.

Na área de engenharia de software, desde o fenômeno Roger Pressman e seu livro Engenharia de Software, que inclusive é uma das referências deste artigo, muitas companhias

passaram a compreender que qualidade apenas pode ser obtida com auditoria do processo e certificação dos itens que possuem maior impacto no resultado do produto.

Abordando especificamente Testes, podemos citar como principais técnicas de testes de software:

- Teste Funcional
- Teste Estrutural
- Teste Baseado em Falha ou Erro
- Teste de Caixa Branca
- Teste de Regressão

4.1. Teste Funcional

O Teste funcional que também é conhecido como teste "caixa - preta" são os testes concentrados de acordo com os requisitos funcionais do software (PFLEEGER, 2004). Como não há conhecimento sobre a operação interna do programa, o analista concentra-se nas funções que o software contemplará. Baseado na especificação determina-se as saídas que são esperadas para um determinado conjunto de dados.

Esse tipo de teste reflete a ótica do usuário, o *stakeholder* interessado em utilizar o programa, sem levar em conta os detalhes de sua construção. Comparado a outros modelos, sua concepção é relativamente simples.

Segundo Koscianski, o teste é particularmente útil para revelar problemas tais como:

- Funções incorretas ou omitidas;
- Erros de interface;
- Erros de comportamento ou desempenho;
- Erros de iniciação e término.

A principal técnica utilizada nos testes funcionais é a de particionamento de equivalência, que visa uma divisão em subconjuntos das entradas de valores de acordo com as funcionalidades do software e escolhendo a melhor abordagem a ser utilizada e obtendo validação dos erros e confiabilidade (BRUNELI, 2006). Como a técnica de particionamento de equivalência existe diversas técnicas que podem ser adotadas como a de Análise do valor limite, Grafo Causa-Efeito e *Error-Guessing* (VINCENZI, 1998) (CORTE, 2006).

4.2. Teste Estrutural

O Teste estrutural ou também conhecido como teste "caixa - branca" ou teste "caixa – de - vidro" ou ainda teste "caixa - clara" são os projetos de casos de testes onde seus testes são desenvolvidos a partir dos conceitos de implementação do software e da estrutura desenvolvida (PRESSMAN, 1995) (PFLEEGER, 2004).

Os testes caixa – branca possibilitam avaliar a estrutura interna do programa, sua codificação, módulos, implementação e execução por partes do programa e de seus componentes elementares, sendo que estas informações também são utilizadas para a criação dos casos de testes (SOMMERVILLE, 2007).

Em geral existem diversos critérios para a elaboração dos testes estruturais que podem ser auxiliados com o apoio dos Grafos de Fluxo de Controle ou Grafo de Programa, mostrando assim um fluxo lógico do programa (VINCENZI, 1998). Os critérios dos testes estruturais podem ser destacados como: Critérios Baseados em Fluxo de Controle, Critérios Baseados na Complexidade e Critérios Baseados no Fluxo de Controle.

Critérios Baseados em Fluxo de Controle: Segue de acordo com as características de execução do programa (comandos e desvios), para a sua elaboração são utilizados critérios que podem ser: Todos – Nós (exige que cada comando da aplicação seja executado pelo menos uma vez, passando por cada vértice do grafo), Todos – Arcos (Requer que cada aresta do grafo seja executada pelo menos uma vez) e Todos– Caminhos (requer que todos os caminhos possíveis do programa seja executado, criando assim um teste exaustivo, pelas inúmeras possibilidades existentes) (VINCENZI, 1998) (CORTE, 2006).

Nós são construídos através do código- fonte, onde os seus comandos são os nós do Grafo, ou seja, são os pontos do grafo, seguindo a sequencia dos comandos; (VILELA c, 2005).

Os Arcos são os desvios que ocorre no código-fonte do programa, as linhas que ligam os nós, ou seja, a transferência de controle entre os blocos; (VILELA c, 2005).

Critérios Baseados na Complexidade: São criados a partir dos requisitos de teste e partir do seu grau de complexidade. Um dos critérios que são utilizados é o Critério de McCabe que visa à execução do programa através de conjuntos lineares independentes, utilizando o conceito da complexidade ciclomática para os requisitos de teste (VINCENZI, 1998) (CORTE, 2006).

A Complexidade ciclomática é uma técnica utilizada para criação de grafos, com a seguinte fórmula CC = Número de arcos – Número de nós + 2, determinando assim a quantidade de casos de teste mínimo para testar os caminhos independentes do programa; (VILELA c, 2005).

Critérios Baseados no Fluxo de dados: Utiliza informações do fluxo de dados do programa para a criação dos requisitos dos testes, neste critério é necessário que o teste de interação seja de acordo com as definições de variáveis e de suas referências (VINCENZI, 1998) (CORTE, 2006).

Na identificação dos caminhos que devem ser testados pode se usar a técnica Complexidade ciclomática, pois através dela é possível fazer a identificação da quantidade de testes necessários garantindo que todas as instruções sejam executadas ao menos uma vez (BRUNELI, 2006).

Na Figura 1, nota-se um grafo de fluxo com auxilio da complexidade ciclomática do código ao lado da figura, onde são separados os estados de acordo com as decisões do programa e obedecendo a sua ordem de execução, sendo que as figuras auxiliam na criação dos casos de teste.

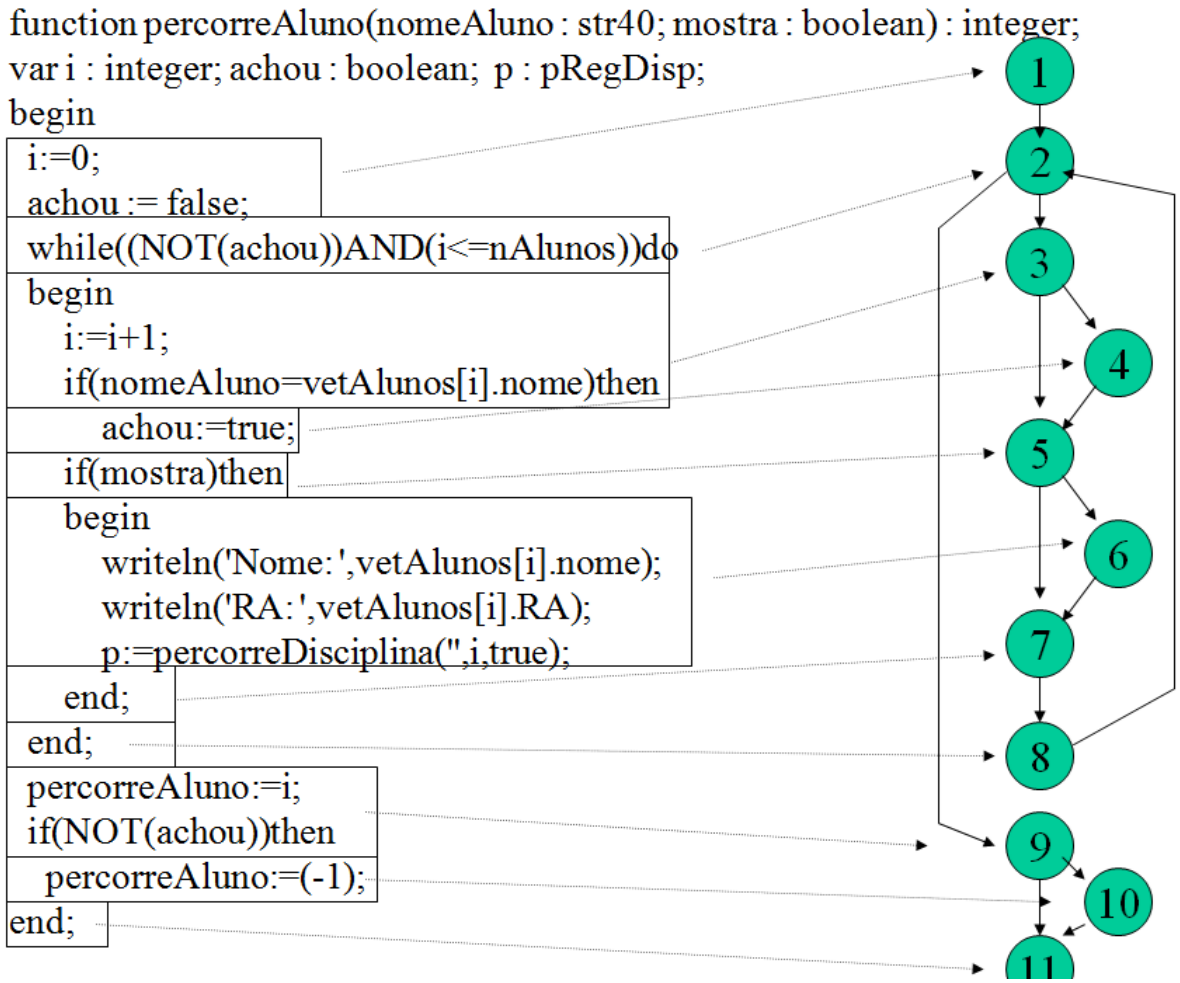

Figura 1 – Exemplo Grafo de Fluxo (VILELA C,2005).

4.3. Teste Baseados em Falha ou Erro

Os Testes Baseado em Erros e Falhas são criado a partir das informações dos erros mais comuns que os desenvolvedores cometem no processo de desenvolvimento, assim utilizados para a criação dos casos de testes (JINO, 2007).

Para o desenvolvimento dos casos de testes, existem algumas técnicas que podem ser empregadas na sua elaboração: Semeadura de Erros e Análise de Mutantes (VINCENZI, 1998).

No critério de Semeadura de Erros são inseridos artificialmente no programa alguns erros. Após estas inserções são analisados os erros encontrados e verificados quais são erros prováveis e improváveis, ou seja, os erros artificiais são descartáveis, através do conceito de probabilidade pode se chegar à quantidade provável de erros naturais que podem conter no software (CORTE, 2006).

No critério de Análise de Mutantes são criados programas com pequenas alterações em cima daquele software que será testado, visando avaliar a quantidade de testes necessários para o software em questão, tendo como foco encontrar casos de testes capazes de revelar diferenças de comportamentos entre o software original e os softwares cujo os trechos foram modificados, ou seja, nos softwares mutantes. Comparando assim os resultados obtidos da versão original e com a versão modificada, chega-se aos erros e aos números esperados de erros (VINCENZI, 1998) (CORTE, 2006).

Os tipos de critérios existentes na Análise de Mutantes são a Mutação Fraca e a Mutação Firme. A Mutação Fraca cria-se o mutante inicialmente, antes da execução e somente no final da execução que os dados são comparados. Na Mutação Firme são realizadas alterações no programa para a criação do programa mutante, sendo que as comparações acontecem antes do final da execução e sim desde inicio até o seu término (VINCENZI, 1998).

4.4. Teste de Regressão

Essa é uma técnica de teste aplicável a uma nova versão de software ou à necessidade de se executar um novo ciclo de teste durante o processo de desenvolvimento. Consiste em se 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. Inclui-se nesse contexto a observação de fases e técnicas de teste de acordo com o impacto de alterações provocado pela nova versão ou ciclo de teste.

Para efeito de aumento de produtividade e de viabilidade dos testes, é recomendada a utilização de ferramentas de automação de teste, de forma que, sobre a nova versão ou ciclo de teste, todos os testes anteriores possam ser executados novamente com maior agilidade.

4.5. Projeto de Casos de Teste

O projeto de casos de testes é o caso que será utilizado como base, tanto para validar os requisitos funcionais quanto para avaliar a eficácia dos componentes.

No Projeto de Casos de Testes são descritas suas entradas e saídas, de acordo com cada característica do software, visando os requisitos e especificações do projeto (PRESSMAN, 1995). Segundo Sommerville, os casos de testes são elaborados para exercitar adequadamente as estruturas do programa.

Os casos de teste podem são utilizados em diversos seguimentos, segundo Sommerville, os principais são:

- Na criação e implementação de esquemas de scripts de testes, fornecendo os pontos–chaves de acordo com as implementações dos requisitos no momento da codificação do software;

- Utilização de scripts para identificação dos melhores testes, tanto para testes manuais quanto para testes automáticos;
- Um controle dos números de teste específicos para abranger todo o software.

Os Testes de Software podem ser dividida00 em etapas como: Teste de Unidade, Teste de Integração, Teste de Componentes, Teste de Validação e Teste de Sistema (PRESSMAN, 1995) (JINO, 2007).

As estratégias de testes permitem a verificação dos erros nos diversos tipos de software, possibilitando uma maior cobertura do software de acordo com o nível do teste, podendo ocorrer em baixo nível ou em alto nível. Os Testes de baixo nível focam uma funcionalidade específica que se deseja testar, enquanto os testes de alto nível têm o objetivo de verificar as funções do software de acordo com os requisitos do sistema (CARDOSO, 2006).

4.6. Teste de Unidade

O Teste de Unidade é a fase de teste que tem como finalidade testar individualmente as funcionalidades do software em questão, garantindo que todas as funcionalidades do software sejam testadas pelo menos uma vez (CARDOSO, 2006).

Uma das desvantagens do Teste de Unidade é que sua implementação é exaustiva e necessita de prazos maiores para realização, sendo que para a realização dos testes deve-se conhecer os objetivos e especificações dos requisitos de cada uma das funcionalidades do software. Enquanto a sua vantagem é a localização e prevenção de falhas por estar testando cada uma das funcionalidades individualmente (PRESSMAN, 1995).

4.7. Teste de Integração

O Teste de Integração é a fase que testa a integração dos componentes do sistema, se estão de acordo com os requisitos do software, levando em consideração a sua funcionalidade em conjunto e não as suas regras de negócios, procurando erros associados a interfaces. O Teste de Integração pode ser dividido em dois tipos (PFLEEGER, 2004) (CARDOSO, 2006): Não Incremental e Incremental.

Não Incremental

Os módulos do sistema são interligados e combinados, e o software é testado como um todo, dificultando a localização dos erros e a correção dos erros encontrados (PFLEEGER, 2004);

Incremental

O software é testado em partes, possibilitando que as interfaces sejam testadas de forma incremental, facilitando encontrar erros e isolá-los para correção. O teste incremental possui duas estratégias: Integração descendente (top-down), onde os módulos do sistema são

integrados de cima para baixo de acordo com a hierarquia de controle do software; e a integração ascendente (bottom-up), onde a construção e os testes dos módulos são iniciados da parte mais baixa da estrutura do software (PFLEEGER, 2004).

4.8. Teste de Componentes

No Teste de Componentes os componentes do software são testados separadamente de acordo com a especificação e estrutura das funcionalidades. Estes componentes são as integrações de interface e unidades do software, com diversas classes no seu desenvolvimento (CARDOSO, 2006).

4.9. Teste de Validação

O Teste de Validação tem como objetivo avaliar se o sistema desenvolvido funciona de maneira que atenda a todas as especificações dos requisitos do software e o processo de regras de negócios estabelecidas na sua elaboração (PRESSMAN, 1995).

4.10. Teste de Sistema

O Teste de Sistema é realizado após o sistema estar pronto, sendo avaliados todos os componentes e funcionalidades. O objetivo do teste de sistema é exercitar todo o software, assegurando que todos os elementos que compõem o sistema estão de acordo com as especificações dos requisitos, incluindo todos os itens de hardware e software que compõem a regra de negócio. Existem vários tipos de teste de sistema, tais como: Teste de Desempenho (agilidade), Teste de Segurança (confiabilidade), Teste de Recuperação (recuperação de dados) e Teste de Inicialização (inserção) (CARDOSO, 2006) (PRESSMAN, 1995).

4.11. Testes Automatizados

Os Testes Automatizados são desenvolvidos como programas ou scripts que têm como principal objetivo exercitar o sistema, testando as funcionalidades e verificando se estão de acordo com as especificações dos requisitos do sistema e os objetivos esperados (BERNARDO, 2008).

Existem diversas vantagens para a utilização dos testes automatizados, dentre elas destacam-se:

- Menor tempo na execução dos testes;
- Verificação do sistema durante o processo de desenvolvimento;
- Alcançar melhor qualidade do software;
- Casos de testes mais elaborados.

A grande vantagem dos testes automatizados é que podem ser repetidos a qualquer momento, seja em uma nova funcionalidade ou alguma modificação em uma funcionalidade específica. Sua implementação reduz os esforços e o tempo gasto, garantindo que não ocorrerá falha humana na execução dos testes (BERNARDO, 2008).

5. AUTOMAÇÃO DE TESTES DE SOFTWARE

Segundo Molinari, existem duas razões básicas para a automatização dos testes:

- O grande crescimento das aplicações e sistemas;
- O elevado grau de complexidade dos sistemas.

Levando em consideração as razões básicas de Molinari, devido ao grande crescimento das aplicações e dos sistemas, existem sistemas que a única solução para a execução dos testes e a utilização a automatização dos testes, levando em consideração o tipo de teste que será realizado e da função a ser realizada, tornando assim algumas vezes a execução do teste manual inviável (MOLINARI, 2008).

Atualmente o analista que desenvolve os testes automatizados, depende da necessidade de entender a programação do sistema desenvolvido e a sua lógica de implementação, sendo antes necessário somente aos programadores do sistema. Há também a necessidade do conhecimento e as características da ferramenta a utilizar na automatização, que muitas vezes possuem linguagens próprias para o seu desenvolvimento (MOLINARI, 2008).

5.1. Tipos de Ferramentas de Automatização de Testes

A classificação dos tipos de ferramentas de automatização de Testes é criada de acordo com cada profissional da área de teste. Abaixo serão demonstradas algumas ferramentas de automatização que terão diversas visões, mostrando o lado de cada profissional, ou seja, visão acadêmica e de mercado (MOLINARI, 2008).

Algumas ferramentas de acordo com o mercado, segundo Molinari:

- Planejamento de Testes: Projeção de testes, criação de requisitos e casos de testes;
- GUI Test Drivers com Command Scripts: Possibilita a gravação e execução das ações dos testes, podendo ser modificadas de acordo com a necessidade do testes, com base scripts com estruturas básicas de programação ;
- GUI Test Drivers com Visual Script: Possui as mesmas funções dos Command Scripts, a única diferença e que não possuem linguagem e sim sendo visual através dos ícones;
- Test Managers: Gestão de automatização e execução dos testes com interface gráfica;
- Static Analysis: Analisa o código do programa sem a necessidade de executar;
- Defect Tracking: Gerenciar um banco de dados com defeitos.

Algumas Ferramentas com base a classificação acadêmica, segundo Molinari:

- Capture / Replay: Captura os comandos executados de acordo com que o usuário está testando e depois executa;
- Debuggers: Retira erros básicos do código;
- Code Auditors: analisa e auditoria o código;
- Test Data Generators: Geração de massa de dados para testes;
- Test Management: Gerenciamento e planejamento de teste.
- Standards Checkers: Verificação de padrões de interfaces.

As ferramentas de automatização de testes existem para diversos fins necessários para a criação de teste, desde seu planejamento até a sua execução, onde as empresas e os *testers* (responsável pela criação da automatização dos testes) verificam a sua necessidade de acordo com os testes que serão executados e qual ferramenta ou ferramentas devem ser utilizadas na sua elaboração e execução dos testes.

5.2. Dificuldades no Processo de Automatização

As dificuldades na automação de testes ocorrem como em todos os processos para a construção de uma aplicação e sistema, que podem ser levando em consideração uma equipe despreparada ou sem experiência, altos custos iniciais e mão de obra não especializada, dificuldade no entendimento dos casos de testes e da lógica de programação da aplicação ou do sistema, que tem grande importância na automatização.

Inicialmente nas empresas a sua principal dificuldade é o processo de implantação dos testes automatizados, pois a automatização dos testes inicialmente tem um elevado custo, devido à compra de ferramentas apropriadas para a criação e execução dos testes, treinamento da equipe, contratação de pessoas qualificadas, e entre outros, sendo que algumas empresas possuem medo de uma grande implantação e gastos com pouco retorno. Essas empresas que possuem este pensamento devem levar em consideração que inicialmente não terão tanto retorno, devido à necessidade de criação dos scripts para os casos de testes e treinamentos, mas em longo prazo obterá grandes retornos, como melhor qualidade em menor tempo, sendo uma característica muito importante para o mercado atual.

Outro grande fator de dificuldade na automatização dos testes é a qualificação das equipes, que deve ter um grande grau que conhecimento das ferramentas e do sistema ou aplicação, para a melhor construção e utilização da automatização, onde diversas vezes e difícil encontrar qualificação e entendimento da aplicação ou sistema em uma única pessoa. Às vezes a pessoa conhece o sistema ou aplicação, mas não conhece o processo de automatização ou vice versa, sendo assim necessário treinamento para uma melhor qualidade e benefícios futuros.

5.3. Critérios de Avaliações

Os critérios de avaliação para a escolha de uma boa ferramenta de automatização de testes, podem ser considerados por diversos fatores e de acordo com a aplicação em que o teste virá a ser aplicado.

Diante das características e tipos de ferramentas de automatização apresentadas, em um contexto geral podem ser considerados os critérios de avaliações como:

- Grau de complexidade da ferramenta;
- Custo da ferramenta;
- Custo de treinamentos e a aprendizagem da aplicação;
- Reutilização do código automatizado do teste;
- Tipo de plataformas a serem utilizadas;

Portanto os critérios de avaliações para uma boa ferramenta de testes devem ser observados de acordo com as características dos projetos empregados nos testes automatizados e de acordo com as características demonstradas.

6. FRAMEWORK CDK

O CDK é um Framework de desenvolvimento do CPqD desenvolvido na linguagem Java que é um kit de desenvolvimento de software, que visa proporcionar um ambiente de desenvolvimento de aplicações mais viável, levando em consideração alguns pontos importantes para o desenvolvimento de aplicações. Alguns pontos são (LELLO, 2008) (OLIVEROS, 2008):

- Custo;
- Prazo;
- Qualidade;
- Robustez;
- Qualidade em um curto espaço de tempo;
- Ambiente de construção crítica.

O principal objetivo do CDK é ser um mecanismo para agilizar no desenvolvimento de aplicações corporativas, buscando alguns pontos - chaves como (LELLO, 2008) (OLIVEROS, 2008):

- Integração de ferramentas para aumentar o controle;
- Comprometimento para aumentar a produtividade;
- Ferramenta de baixo custo, visando não aumentar os custos indiretos;
- Construção de uma arquitetura robusta para sistemas independentes de seus fornecedores.

6.1. Composição

O Framework CDK é composto por um conjunto de componentes na sua infra-estrutura como: Controle de acesso, Montagem de Interface no padrão do CPqD, Controle de licenças, Administração de Banco de Dados e entre outros componentes, que auxiliam inicialmente para a construção de uma aplicação, estes componentes permitem que o desenvolvedor se preocupe com as regras de negócios e não com a infra-estrutura da aplicação, gerando assim uma maior produtividade e padronização na forma de desenvolvimento (LELLO, 2008) (OLIVEROS, 2008).

A composição do CDK é:

- Processos: CDK Control (processos);
- Ferramentas: CDK Tools (ferramentas que auxiliam nos desenvolvimentos);
- Componentes: CDK Framework (componentes que estão no Framework).

CDK Controls

O CDK Control são os processos que ocorrem dentro do CDK, demonstrando todos os processos que ocorrem dentro do processo de desenvolvimento de uma aplicação (LELLO, 2008) (OLIVEROS, 2008).

CDK Tools

O CDK Tools são as ferramentas que auxiliam para o desenvolvimento da aplicação, dando assim apoio na sua criação, são divididas em ferramentas administrativas, ferramentas de desenvolvimento, ferramentas utilitárias e ferramentas corporativas (LELLO, 2008) (OLIVEROS, 2008).

CDK Framework

O CDK Framework é onde estão todos os componentes utilizados no CDK. Estes componentes são divididos em três níveis (LELLO, 2008) (OLIVEROS, 2008):

- Componentes Tecnológicos;
- Componentes de Negócios;
- Componentes de Solução.

7. ESTUDO DE CASOS DE TESTES FUNCIONAIS

O Framework do CDK (CPqD Development Kit) foi utilizado como base na criação de uma aplicação Teste cuja aplicação, é para demonstrar a execução dos testes automatizados funcionais e os scripts de automatização dos testes, demonstrando os resultados obtidos e esperados na execução de funcionalidades simples como inserir dados na aplicação.

A aplicação Teste possui interações para a criação de mapas, com o auxílio da ferramenta AutoCad Map 3D 2008 e a aplicação utilizada para demonstrar os testes foi desenvolvida na linguagem Java.

A Figura 2 demonstra a tela inicial da aplicação Teste que será utilizada para a execução dos testes automatizados.

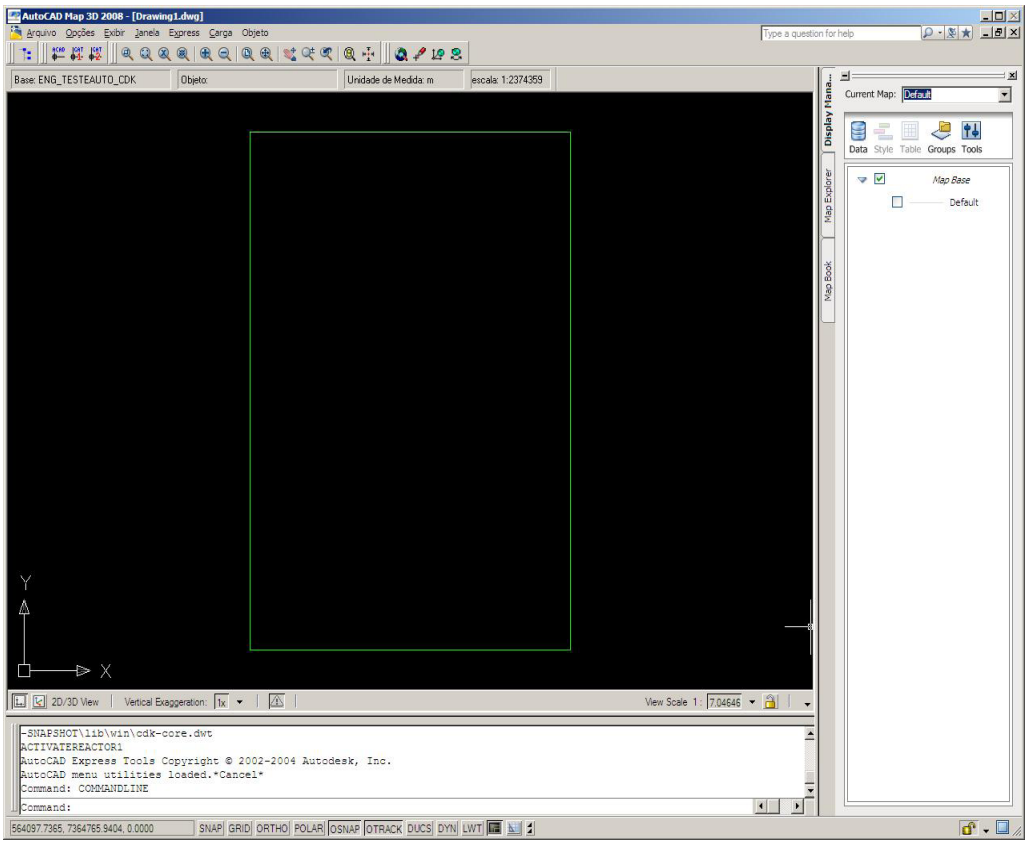

Figura 2 – Tela inicial da aplicação Teste.

7.1. Técnicas utilizadas

Os testes funcionais são considerados exaustivos, por testar as funcionalidades do software de acordo com os seus requisitos, fornecendo entrada de valores ao sistema que serão avaliadas e verificadas se estão de acordo com as conformidades esperadas do software (VILELA B, 2005). Devido a grande quantidade de valores que podem ser levantados para a realização dos testes foram criadas técnicas para facilitar a sua utilização. As técnicas utilizadas no desenvolvimento do trabalho foram a de Particionamento e Equivalência e Análise de Valor Limite.

Técnica de Particionamento e Equivalência

A técnica de particionamento de equivalência visa diminuir a exaustão dos testes, devido ao grande número de entradas de valores que podem conter em cada funcionalidade do software (JINO, 2007) (VINCENZI, 1998). Na definição dos casos de testes serão criados subconjuntos das entradas de valores principais, de acordo com a funcionalidade do

software juntamente com as suas equivalências. O foco principal dos subconjuntos é que o elemento das classes que se comporta de maneira similar em relação a outros elementos da mesma classe ou de outras classes, terá os mesmo casos de testes, reduzindo assim as quantidades de avaliações de entradas e de saídas de valores (CORTE, 2006).

Para a definição das classes de equivalência devem ser levadas em consideração algumas condições para a sua construção (BRUNELI, 2006):

- Condições de entrada com intervalo de valores;
- Condições de entrada com quantidades limitadas de valores;
- Condições de entrada com conjunto de valores válidos e com manipulações diferentes;
- Condições de entrada com incerteza de valores.

Depois de identificados as classes de equivalência devem se determinar os casos de testes, escolhendo os elementos chaves (que possuem equivalência com outros elementos), procurando uma maior cobertura do teste em questão.

Técnica de Análise Valor Limite

A técnica de analise do valor limite verifica a manipulação dos dados de entrada de valores nos seus limites (mínimos e máximos), podendo ser considerado um complemento da técnica de particionamento de equivalência. Os testes com valores limites são muito importantes, pois são nestes pontos que existe uma maior concentração e probabilidade de erros (VINCENZI, 1998) (CORTE, 2006).

Ferramenta de Automatização

A automação dos testes funcionais descritas no presente trabalho, foi desenvolvida com base no Software IBM Rational Suite, utilizando plataforma Rational Robot, mostrando a ferramenta, seus benefícios e demonstrando o processo de desenvolvimento dos scripts.

O Rational Robô é uma ferramenta para automação de teste funcional e regressiva. No Rational Robô são desenvolvidos scripts para executar as funcionalidades do sistema que deseja testar, podendo criar logs com as informações dos testes realizados, também possibilita a gravação da execução passo a passo do teste que pode ser implementado para melhor atender as necessidades, assim após a gravação pode – se executar diversas vezes aquela funcionalidade do sistema. Robot também permite aos usuários criar *datapools* , conjuntos de dados de ensaios que possam ser utilizados pelos scripts automatizados para variar os valores de entrada na aplicação durante a execução dos scripts (IBM, 2009).

7.2. Desenvolvimento dos Scripts dos Testes Automatizados

Os scripts de automatização foram desenvolvidos através da ferramenta da Rational Robot, onde nesta unidade será demonstrado o desenvolvimento dos scripts automáticos de testes funcionais, tendo como base a aplicação Testes, utilizando o CDK.

Inicialmente na automatização dos testes é utilizado o processo Record, que possibilita a gravação da utilização da aplicação que deseja ser testada, sendo que os scripts podem ser incrementados e reproduzir diversas vezes para analisar a aplicação se está funcionando corretamente de acordo com os requisitos do projeto.

O desenvolvimento dos scripts é a partir da linguagem SQABASIC , linguagem especifica da ferramenta Rational Robot, tendo como base a linguagem ASP .

A automatização dos testes através de scripts podem ser divididos em 4 fases:

- Gravação dos Scripts;
- Implementação dos scripts, tendo como base a gravação dos scripts, ou seja, adicionar funcionalidade como, por exemplo: arquivos de logs, utilização do *datapools*, entre outros;
- Execução dos scripts.
- Verificação dos resultados da execução e os arquivos de logs, caso tenham sido implementados.

7.3. Execução dos Testes

A execução dos Testes tem como finalidade a avaliação das vantagens e desvantagens dos scripts de automatização do Teste funcional, para inserção de dados da aplicação do estudo de caso, verificando ainda se terá o comportamento esperado e se as inserções ocorreram com sucesso, caso ocorra algum erro na funcionalidade ou nas inserções dos atributos.

Na execução dos Testes terão como base a função inserir do objeto Linha da aplicação Teste (estudo de caso) utilizando o CDK na plataforma desktop, com interação com o AutoCad. Ao final da execução devem ter inserido de acordo com a quantidade de dados do *datapool* a mesma quantidade de linha na aplicação.

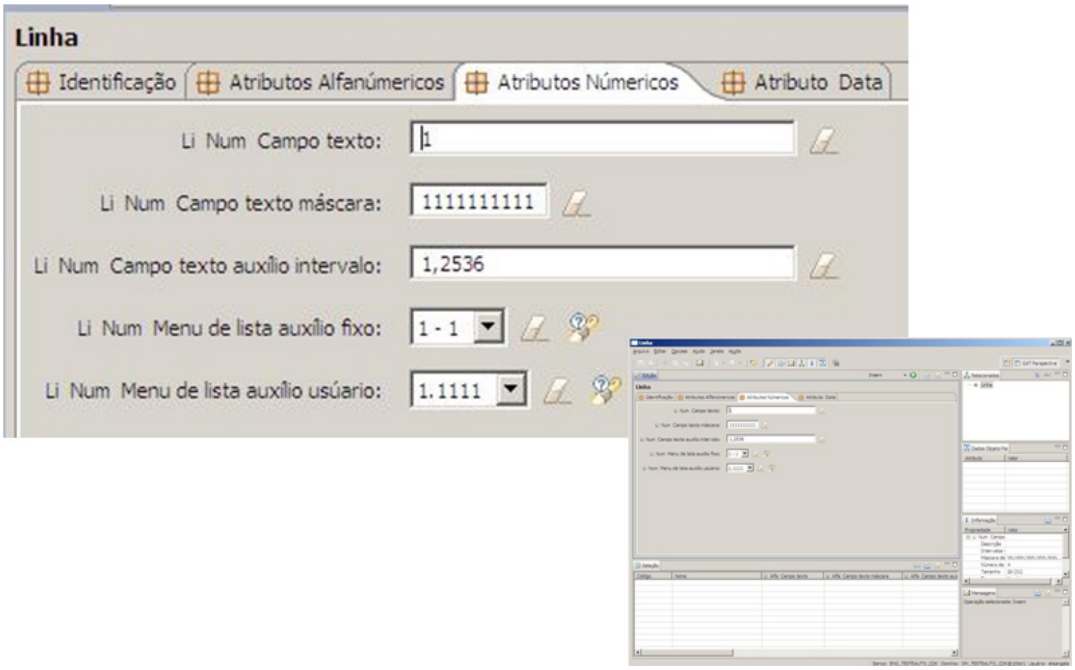

Figura 3 – Tela de inserção dos atributos numéricos.

A Figura 3 demonstra a inserção nos atributos numéricos de acordo com seus tipos (EditBox, Área de Texto, etc) para o objeto Linha da aplicação Teste.

A inserção das linhas será repetida dentre a quantidade de vezes que conter informações no *datapool* referente a execução desta funcionalidade do software, testando assim a maior quantidade de opções de inserções e combinações possíveis nos atributos, buscando a validação e verificação das conformidades da funcionalidade de inserir do objeto Linha da aplicação Teste (estudo de caso).

A Figura 4 demonstra um exemplo de varias inserções de linha na aplicação Teste após a execução dos scripts automático, que estão demonstradas através da delimitação retangular da Figura 4.

Portanto, as execuções dos testes modificam de acordo com a funcionalidade testada da aplicação, foi utilizada como exemplo a inserção do objeto Linha da aplicação Teste (estudo de caso) para demonstrar de forma prática o comportamento inicial, intermediário e final da inserção com o script automatizado para o melhor entendimento da execução dos testes automatizados.

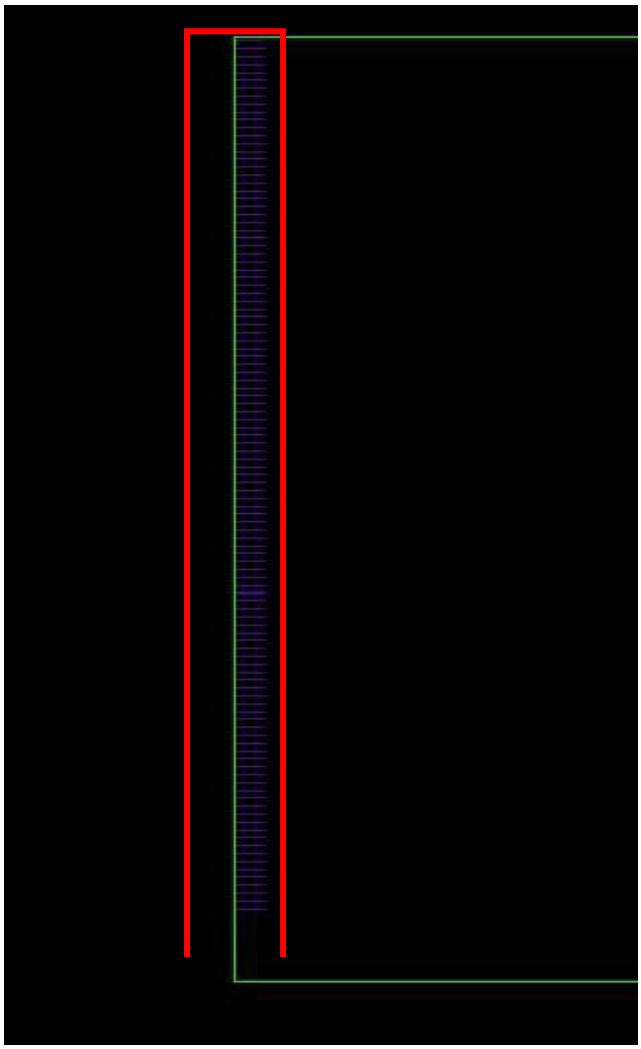

Figura 4 – Tela com diversas inserções de linhas.

7.4. Análise Tempo de execução

O tempo de execução de um script automatizado leva em consideração a funcionalidade a ser testada e a quantidade de informações a serem inseridas no software. Este tópico tem como finalidade demonstrar a comparação do tempo de execução da inserção do objeto Linhas, conforme utilizado como exemplo na aplicação Teste (estudo de caso), a sua comparação é feita com a inserção manualmente e com a inserção utilizando o script automatizado.

Segue abaixo uma tabela com os dados da execução tanto manual como automatizado para o Objeto Linha do estudo de caso:

Tabela 1 – Comparação de tempo de execução de inserções.

Quantidade de Inserções de Linhas	Tempo de execução com Inserção Manual	Tempo de execução com Inserção utilizando script automatizado
1 Linha	1 Minuto e 15 Segundos	0 Minutos e 36 Segundos
10 Linhas	12 Minutos e 30 Segundos	06 Minutos e 00 Segundos
. . .	. . .	. . .
110 Linhas	2 Horas e 17 Minutos e 30 Segundos	1 Hora e 06 Minutos e 00 Segundos

Na Tabela 1 demonstra os dados que possibilitam a comparação do tempo de execução nas inserções de Linhas manualmente e com os scripts automatizados.

Com os dados da tabela podemos notar claramente o beneficio dos testes automatizados em relação ao tempo de execução da inserção manual da funcionalidade inserir do Objeto Linha da aplicação Teste (estudo de caso), possibilitando assim que os testes sejam executados em menor tempo e menores prazos de entrega, com verificações e validações mais complexas e verificando na execução os erros e falhas que podem ocorrer na funcionalidade do software testada.

7.5. Resultados Obtidos

Os resultados obtidos foram analisados a partir do estudo de caso proposto, sendo baseados somente na execução do teste e não comparando custos para a automatização, tempo para a criação dos casos de uso e do tempo para a criação dos scripts automatizados dos testes funcionais da aplicação Teste.

O foco principal dos resultados é a verificação da funcionalidade do software de acordo com o estudo de caso proposto e de acordo com os requisitos do sistema e validações dos atributos utilizados no Objeto Linha.

As avaliações dos resultados diante do contexto proposto da execução e verificação e validação da funcionalidade do Objeto linha foram:

- Otimização do tempo de execução;
- Grande quantidade de interações para verificação das validações dos atributos nas inserções;
- Verificação dos erros e falhas encontrados na execução;
- Grande quantidade de inserções no mapa, verificando o seu comportamento de acordo com as inserções das linhas;

Diante dos resultados propostos vamos analisados isoladamente, verificando os seus resultados.

Na otimização do tempo de execução obtivemos um grande avanço, sendo que os testes funcionais automatizados levam a metade do tempo para serem executas em comparação com os testes manuais como demonstrado na Tabela 1. O tempo tem grande importância nos testes, pois o quanto antes testar o caso de uso proposto, mais rápido serão encontrados os erros e falha e corrigidos, onde em alguns momentos com a ótima realização da atividade de teste com os scripts automático quando liberados ao cliente para a utilização, diminuíram a chance de encontrar erros e falhas, ocasionando assim grande satisfação do cliente e maior qualidade do software perante o cliente e a sociedade.

Com a otimização do tempo de execução é possível à utilização grande quantidades de interações nas inserções do Objeto Linha, assim podendo avaliar as validações dos atributos com inúmeras possibilidades, encontrando assim os erros e falhas mais facilmente.

Nas verificações e validações dos atributos da funcionalidade inserir do Objeto Linha podem-se encontrar alguns erros ou falhas no software na execução deste caso de testes, utilizado como estudo de caso. Com a utilização da automatização dos testes com arquivos de *logs* pode encontrar facilmente o local do erro e a que momento ele aconteceu, facilitando assim a sua correção. Segue abaixo uma falha encontra na execução do caso de teste do Objeto Linha, sendo este um das falhas e erros possível, usando somente para uma visão pratica de como ocorre uma falha no software.

8. CONSIDERAÇÕES FINAIS

Para dar continuidade a este trabalho sobre automatização de testes funcionais, podemos identificar as seguintes propostas:

- Adaptação dos métodos dos scripts de automatização dos testes, assim como o processo de desenvolvimento;
- Melhoria dos scripts para execução de outros tipos de testes, assim como, teste de regressão, de exaustão, de confiabilidade e entre outros;
- Aplicação em outros estudos de caso, para a medição de sua eficiência;
- Reprodução da eficiência da automatização dos testes em outras aplicações para obter informações mais abrangentes sobre a sua eficiência;
- Utilização dos princípios e base da proposta do trabalho para aperfeiçoamento de futuras ferramentas e outro framework em qualquer outra aplicação.

REFERÊNCIAS

BERNARDO, Paulo C.; KON, Fabio, A importância dos Testes Automatizados. Artigo Revista Engenharia de Software Magazine, página 54-57, 2008.

BORLAND. Borland SilkTest. Artigo Disponível http://www.borland.com/br/products/silk/silktest/index.html, Site Borland, Acesso em: 29 de Agosto de 2009.

BRUNELI, Marcus V. Q., A utilização de uma metodologia de Teste no processo de melhoria da Qualidade de Software. Dissertação de Mestrado, Unicamp, Campinas, São Paulo, Fevereiro, 2006.

CARDOSO, Josiane Ap., Um método de Testes de Integração para Sistemas Baseados em Componentes. Dissertação de Mestrado, Unicamp, Campinas, São Paulo, Fevereiro, 2006.

CORTE, Camila K. D., Ensino integrado de fundamentos de Programação e Teste de Software. Dissertação de Mestrado, ICMC – USP, São Carlos – São Paulo, Maio, 2006.

COSTA, Eudes, Ferramentas de Teste: Selenium, São Paulo – São Paulo, Fevereiro, 2008

FEOFILOFF, Paulo, O que é um Grafo? .São Paulo, São Paulo, IMEI - USP, 2005, Notas de Aula.

GUIMARÃES, José O., Frameworks. Artigo Departamento de Computação UFSCar, UFSCar, São Carlos, São Paulo, Novembro, 2000.

IBM. Rational Robot. Artigo Disponível <http://www-142.ibm.com/software/dre/ecatalog/detail.wss?locale=pt_BR&synkey=K108274U58759I63> , Site IBM, Acesso em: 19 de Março de 2009.

JINO, Mário; MALDONADO, José C.; DELAMARO, Márcio E. , Introdução ao Teste de Software. São Paulo: Campus Elsevier, 2007.

LOZANO, Fernando, Software Livre Para o Desenvolvedor, Artigo Disponível http://augustocampos.net/revista-do-linux/023/desenvolvimento.html, Site Revista do Linux, Acesso em: 10 de Setembro de 2009.

LELLO, João D., Documento de Arquitetura – CDK. Documentação CDK, Março, 2008.

MACEDO, Michel P.; DESCHAMPS, Alexandro, Framework, Campinas – São Paulo, 2008.

MOLINARI, Leonardo, Testes Funcionais de Software. Florianópolis: Visual Books, 2008.

NOGUEIRA, Elias, Conhecendo o BadBoy. São Paulo – São Paulo, Janeiro, 2008.

OLIVEROS, Dario, Portal CDK – Guia de Referência. Documentação CDK, Última atualização Outubro, 2008.

PFLEEGER, Shari L. Engenharia de Software: Teoria e Prática. 2ª ed. São Paulo: Pearson Prentice Hall Brasil, 2004.

PRESSMAN, Roger S. Engenharia de Software. São Paulo: Pearson Makron Books, 1995.

REINALDO, Francisco A. F., Definição e Aplicação de um Framework para desenvolvimento de redes Neurais Modulares e Heterogêneas. Dissertação de Mestrado, Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, 2003.

RICARTE, Ivan L. M., Teste de Software Orientado a Objetos. Campinas, São Paulo, DCA – FEEC – Unicamp, 2006, Notas de Aula.

SOMMERVILLE, I. Engenharia de Software. 8ª ed. São Paulo: Pearson Education Brasil, 2007.

VILELA, Plínio, Introdução ao teste de Software. Piracicaba, São Paulo, Unimep, 2005, Notas de Aula. (A)

VILELA, Plínio, Teste de Software: Técnicas Estruturais. Piracicaba, São Paulo, Unimep , 2005, Notas de Aula. (B)

VILELA, Plínio, Teste de Software: Técnicas Funcionais. Piracicaba, São Paulo, Unimep , 2005, Notas de Aula. (C)

VINCENZI, Auri M. R., Subsídios para Estabelecimento de Estratégias de Teste Baseado na Técnica de Mutação. Dissertação de Mestrado. USP, São Carlos – SP, 1998.