doi: 10.5102/un.gti.v2i2. 1956

Ferramentas free para teste de software: um estudo comparativo*

Free tools for software testing: a comparative study

Wanessa Mariana Silva[1]
Angélica Toffano Seidel Calazans[2]

Resumo

O presente trabalho tem por objetivo identificar ferramentas *free* para teste de *software*, analisar e contextualizar sua atuação no processo de teste. Para isso, demonstrou-se a importância do ato de testar para garantir a melhor qualidade de um *software* e conceituou o processo de teste e suas fases. Identificaram-se algumas ferramentas gratuitas existentes no mercado, suas características e funcionalidades em um estudo comparativo. O estudo adotou uma abordagem qualitativa e, como instrumentos de coleta de dados, utilizou a pesquisa documental e questionários semiestruturados para obter a percepção de técnicos de teste sobre as ferramentas. O método adotado foi o estudo de caso e, para a análise dos dados, foi utilizada a técnica de análise de conteúdo. A pesquisa identificou que, das ferramentas estudadas, algumas possuem estudos sobre sua aplicabilidade. O estudo demonstrou que, segundo a percepção dos participantes, parte das ferramentas avaliadas possui recursos eficientes e atende aos objetivos propostos.

Palavras–chave: Teste. Ferramentas de testes *free*. Processo de teste.

Abstract

This study aims to identify tools for free software testing, analyze and contextualise their performance in the testing process. To do this, demonstrated the importance of the act of testing to ensure the best quality and conceptualized the software testing process and its phases. We identified some free tools available on the market, their features and functionalities in a comparative study. The study adopted a qualitative approach, and as instruments for data collection, used the documentary research and semi-structured questionnaires for the perception of technical test on the tools. The method adopted was the case study and to the data analysis, we used the technique of content analysis. The research identified that the tools studied, few studies have applicability. The study showed that, according to the participants' perceptions of the tools evaluated has powerful features and meets the objectives.

Keywords: Test. Testing tools free. Testing process.

* Artigo recebido em 25/06/2012
Aprovado em 27/08/2012

[1] Estudante universitária na instituição Centro Universitário de Brasília - Uniceub, cursando o terceiro semestre do curso de Análise e desenvolvimento de sistemas. Participante do Projeto de Iniciação Científica-PIC.

[2] Doutorado em Ciência da Informação pela Universidade de Brasília (2008) e mestrado em Gestão do conhecimento e TI pela Universidade Católica de Brasília (2003). Atuou 28 anos como especialista da Caixa Econômica Federal e atualmente é professor titular do Centro Universitário de Brasília-UniCEUB.

1 Introdução

A construção de um produto de *software* é uma tarefa complexa. O nível de complexidade depende das características e dimensão do sistema a ser criado. A qualidade de um produto de *software* está relacionada a vários fatores: atendimento das necessidades explícitas e implícitas do cliente, confiabilidade, segurança, portabilidade etc., mas também está fortemente relacionada a existências de defeitos. Parte desses defeitos pode ser corrigido por meio das atividades de teste.

Segundo Delamaro et al (2007), as atividades de teste de *software* têm a finalidade de garantir que o produto esteja em conformidade com o que foi especificado. Pontes (2009) complementa essa informação citando que as atividades de teste podem ser responsáveis por uma parcela considerável dos custos de um projeto. Portanto, merecem muita atenção por parte das organizações que desenvolvem produtos de *software*. Esse autor ressalta que as atividades de teste serão mais bem desempenhadas se estiverem vinculadas a um processo.

Nos últimos anos, a evolução tecnológica contribuiu para o aparecimento de várias ferramentas de automação de testes *free* e pagas. A automação desse processo é essencial, considerando a necessidade de agilidade e a redução de custos das organizações. As ferramentas *free* existentes têm os mais diferentes objetivos e funcionalidades: algumas automatizam um tipo de teste, outras, parte do processo de teste.

Conhecer suas funcionalidades, seu objetivo dentro do processo de teste, suas vantagens e desvantagens é importante para que, tanto as organizações como o meio acadêmico, possam adotá-las de forma eficiente e produtiva.

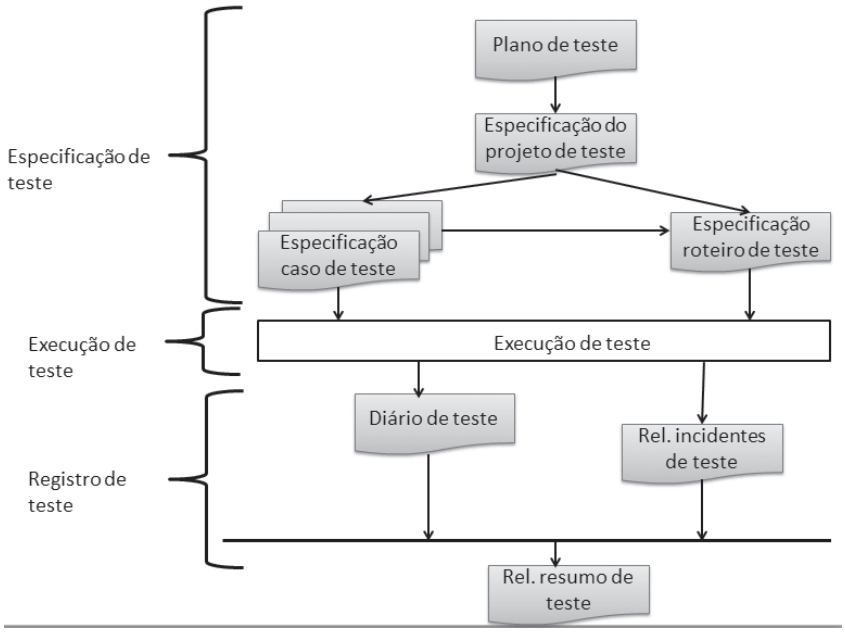
Assim, nas seções seguintes, serão apresentadas uma breve descrição do processo de teste e tipos de teste (Seção 2) e, uma descrição sucinta da importância da automação do processo de teste (Seção 3). Na Seção 4, apresenta-se a metodologia adotada para este trabalho. Na Seção 5, demonstra-se a aplicação da pesquisa e descrevem-se os resultados da pesquisa e do estudo de caso. Finalmente, na Seção 6, serão apresentadas as conclusões.

2 Processo de teste

Um processo de teste pode ser entendido como um processo utilizado para revelar defeitos em um *software* e para estabelecer se ele alcança um determinado nível de qualidade. Assim sendo, o processo de teste é um componente vital em um processo de qualidade e uma das atividades mais desafiadoras e custosas cumpridas durante o desenvolvimento e manutenção de um produto de *software*.

O processo de teste engloba uma estruturação de etapas, atividades, artefatos, papéis e responsabilidades que buscam a padronização dos trabalhos, gerenciar e monitorar os projetos de testes. O padrão IEEE 829 para documentação de testes de *software* especifica a forma de uso de um conjunto de documentos em oito estágios definidos de teste de *software*; cada estágio potencialmente produzindo seu próprio tipo de documento, conforme a Figura 1.

Figura 1 - Padrão 829 para Documentação de Teste de Software



Fonte: IEEE 829

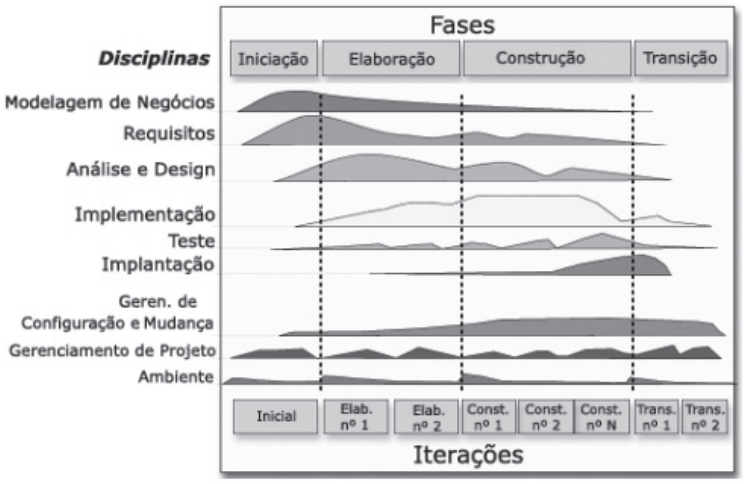
O Plano de Testes, segundo esse padrão, contém os objetivos e metas globais do teste, enquanto o Projeto de Testes detalha e especifica como o Plano de Testes será executado. O caso descreve as situações que devem ser testadas e o Roteiro de Teste cita as ações que deverão ser executadas no *software* para que o Caso de Testes seja executado.

Já o Registro (ou evidência) de Teste descreve os testes executados, independentemente de terem sido encontrados erros ou não, enquanto o Relatório de Incidente descreve as falhas ocorridas durante a execução dos testes e, finalmente, o Relatório Resumo (ou executivo) de Testes contém o resumo das condições de teste executadas, defeitos encontrados e tabulações estatísticas desejadas.

Testar um *software* é mais abrangente do que relatar impressões e não conformidades. *A Rational Software* (2008) apresenta no *Rational Process Unified* (RUP) as várias disciplinas envolvidas no processo de desenvolvimento de *software*. O RUP é um processo proprietário de desenvolvimento de *software* criado pela *Rational Software Corporation* (adquirida pela IBM).

O RUP é um modelo iterativo e incremental que apresenta técnicas e práticas aprovadas comercialmente e utiliza a abordagem da orientação a objetos e a notação *Unified Modeling Language* (UML) para ilustrar os processos. Nesse modelo, a disciplina de testes, conforme pode ser observada na Figura 2, permeia todas as fases de desenvolvimento de *software*.

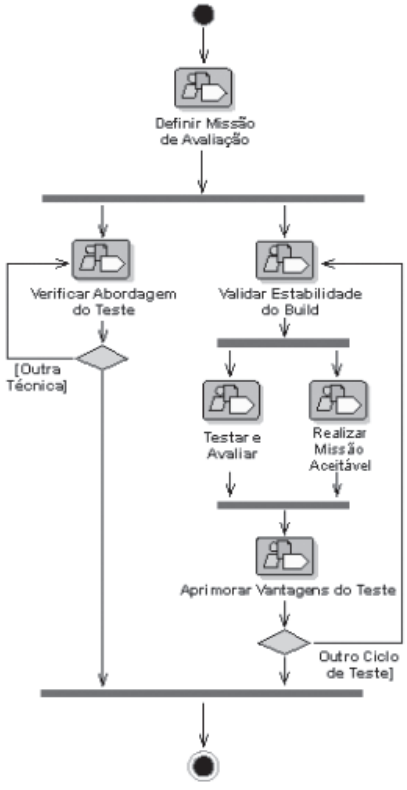
Figura 2 - Fases de desenvolvimento



Fonte: (RATIONAL SOFTWARE, 2008)

A expansão da disciplina de teste, no modelo RUP apresenta um fluxo de trabalho, demonstrado na Figura 3, com as seguintes atividades: definir missão, verificar abordagem de teste, validar estabilidade do *build* (construção), testar e avaliar, realizar missão aceitável e aprimorar vantagens do teste. Tais atividades serão descritas sucintamente a seguir.

Figura 3 - Fluxo de trabalho de teste



Fonte: (RATIONAL SOFTWARE, 2008)

O processo de "Definir missão" tem como objetivo identificar o foco do esforço de teste para a iteração e estabelecer o consenso entre as pessoas envolvidas sobre as metas do esforço de teste. Já o processo de "verificar a abordagem de teste" tem como meta demonstrar quais técnicas facilitarão o teste exigido. O objetivo é compreender as restrições e limitações de cada técnica e encontrar uma solução de implementação adequada ou mesmo, encontrar técnicas alternativas que possam ser implementadas.

A finalidade do fluxo "Validar estabilidade do *build*" é tornar o *buid* estável o suficiente para iniciar o teste e o esforço de avaliação. Esse trabalho também é chamado de "teste de regressão", "teste de verificação do *build*", "teste de regressão do *build*", "verificação de limpeza" ou "aceitação no teste". Esse trabalho ajuda a economizar recursos de teste em esforço inútil. Já o processo de trabalho "testar e avaliar" objetiva atingir a amplitude e o detalhamento apropriados do esforço de teste para permitir uma avaliação suficiente do item de Teste-Alvo. Normalmente realizado uma vez por ciclo de teste, esse trabalho envolve a execução do trabalho de teste e avaliação, ou seja, a implementação, a execução e a avaliação de testes específicos e o relatório correspondente dos incidentes encontrados.

O objetivo do processo de "Realizar missão aceitável" é oferecer um resultado útil de avaliação. Nesse trabalho, prioriza-se ativamente o conjunto mínimo de testes necessários que deve ser aplicado, defendendo a resolução de problemas importantes que têm um grande

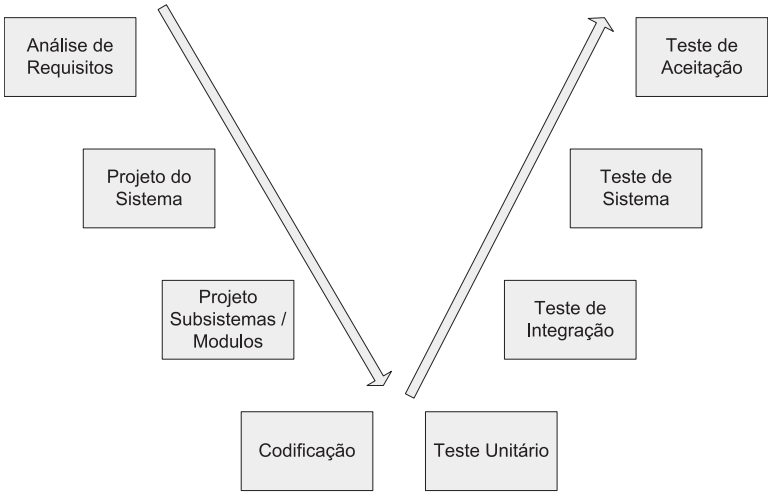
impacto negativo e a qualidade adequada. Identificam-se também as regressões de qualidade introduzidas entre os ciclos de testes e, se pertinente, reavalia-se a Missão. A finalidade do processo "Aprimorar vantagens do teste" é manter e melhorar as vantagens do teste.

Além do processo de teste definido pelo RUP (IBM), o modelo em "V" de teste de *software* (PFLEEGER, 2004) enfatiza as atividades de validação e verificação com o intuito de prevenir/detectar defeitos e minimizar os riscos do projeto. Verificação relaciona-se aos testes aplicados em documentos gerados em cada fase do desenvolvimento. Cada etapa deve produzir um conjunto de documentos, e é possível estabelecer a qualidade por meio da qualidade dos documentos produzidos.

Já a validação avalia a qualidade do produto, que pode ser garantida mediante sistemáticas aplicações de testes nos vários estágios de desenvolvimento da aplicação. Por exemplo, quando se constrói uma unidade de *software*, valida-se a sua estrutura interna e sua aderência aos requisitos estabelecidos ou mesmo a integração de um módulo com as demais unidades.

Para cada fase do processo de desenvolvimento do *software*, o modelo em "V" introduz uma fase, ou nível de teste correspondente. Nesse modelo, o planejamento e a especificação dos testes ocorrem de cima para baixo, ou seja, ao longo das fases de desenvolvimento de *software,* os testes são planejados e especificados. A execução dos testes ocorre no sentido inverso, como pode ser observado na Figura 4.

Figura 4 - Modelo em V para desenvolvimento de software



Fonte: (Adaptado de Pfleeger, 2004)

Complementando essas definições, Caetano (2008) cita a existência de duas técnicas de teste. A técnica do Teste Estrutural, conhecida como "Teste de Caixa Branca", no qual são usados critérios para a geração de casos de teste com o objetivo de identificar defeitos nas estruturas internas do *software*. Enquanto que na técnica do Teste Funcional, também conhecida como "Teste de Caixa Preta", são utilizados critérios para a geração de casos de teste com o objetivo de avaliar a aderência, ou conformidade do *software* implementado em relação ao comportamento descrito nos requisitos.

Além dessas técnicas, diversos autores como Sommerville (2010) e Pressman (2005), identificam vários tipos de teste: funcionalidade, usabilidade, performance, segurança, regressão, carga e configuração, entre outros, que são detalhados a seguir.

2.1 Tipos de teste

Segundo Pressman (1995, p. 45), os testes podem ser classificados em:

- Teste de unidade – concentra-se no esforço para verificação da menor unidade de software. Seu principal objetivo é encontrar falhas de um funcionamento dentro de uma pequena parte do sistema funcionando. O teste de unidade faz muito o uso das técnicas de teste de caixa branca, exercitando caminhos específicos da estrutura de controle. Uma vantagem dessa modalidade é a descoberta de falhas mais cedo no processo de desenvolvimento, trazendo assim, consequências positivas ao longo do projeto.

- Teste de integração – é uma técnica que descobre erros associados a interfaces. O seu objetivo é garantir que os módulos testados no nível de unidade, funcionarão quando integrados.
- Teste de validação – é realizado por meio de uma série de testes de caixa preta que demonstram a conformidade com os requisitos. As duas saídas possíveis do teste de validação são: conformidade do resultado com as especificações ou desvio das especificações com criação de lista de deficiências.
- Teste de sistema – engloba uma série de testes, cujo propósito primordial é pôr completamente a prova o sistema. O teste de sistema pode englobar teste de recuperação, segurança, estresse e desempenho;
- Teste de recuperação – é o teste que força o sistema a falhar de diversas maneiras e verifica se a recuperação foi adequadamente executada;
- Teste de segurança – verifica se todos os mecanismos de proteção embutidos no sistema estão de fato protegendo o sistema de acessos indevidos, ataques etc.
- Teste de estresse – é executado para verificar o comportamento do sistema quando se exige do mesmo quantidade, frequência ou volume anormais.
- Teste de desempenho – é utilizado para testar o desempenho do software, considerando o software e suas integrações. Frequentemente é combinado ao teste de estresse.

3 A importância da automação do processo de teste

Segundo IEEE Std 610, teste é "[...] o processo de operar um sistema ou componente sob determinadas condições, observando ou registrando resultados, realizando a avaliação de algum aspecto do sistema ou componente".

A necessidade da entrega cada vez mais rápida de produtos de *software* faz com que o processo de teste de *software* necessite constantemente de rapidez e agilidade. Assim sendo, a automação tem sido constantemente utilizada para suprir essa necessidade. É importante ressaltar que a diferença entre testes e automação de testes é que o primeiro termo refere-se ao ato de testar, já automação é utilizar um *software* para imitar a interação do ser humano com a aplicação a ser testada.

Embora a automação, quando com foco na execução de testes, exija um tempo maior de elaboração dos casos de teste e o acréscimo de tempo para a criação dos *scripts* de teste, é importante ressaltar que a automação evita o retrabalho, pois facilita o teste de regressão. Viabiliza também a execução de testes em telas já homologadas e garante, dessa forma, que as funcionalidades testadas não apresentam falhas após a implantação de novas funcionalidades.

Uma vez gerado o *script*, é possível:

- Validar vários casos de teste;
- Realizar testes de regressão quantas vezes forem necessárias;
- Alterar os scripts reaproveitando-os para a elaboração de novos testes automatizados (desde que a ferramenta permita);
- Garantir que os passos foram seguidos, como nos testes anteriores.

As ferramentas para automação, nesse contexto, oferecem diversas formas de gerar *scripts*, sendo necessário ou não, conhecer alguma linguagem de programação. Na visão de Molinari (2010), a análise das ferramentas de automação de testes deve ser realizada considerando o processo de desenvolvimento, pois existem ferramentas de teste para diversas dimensões do processo de teste, desde controlar os defeitos, gerenciar o processo de teste, a execução dos testes etc.

Conforme afirma Caetano (2010), a automação de testes tem se tornado vital em projetos de testes de *software,* e a vasta quantidade de opções comerciais e *Open Source* são razões motivadoras para essa mudança, levando em conta as aplicações cada vez mais complexas e orçamentos curtos, tornando assim as ferramentas *Open Source* mais atraentes dentre várias opções existentes.[1]

Para Hayes (2009), a evolução das ferramentas de automação de testes, nos últimos 25 anos, pode ser dividida em 5 fases:

1. Ferramentas de captura de texto e *playback*;
2. Ferramentas de captura de texto e *playback* com interface *caracter*;

[1] O movimento pelo código aberto (*Open source*) foi iniciado nos finais dos anos 90. Contudo, a maioria dos *softwares open source* são considerados *softwar*e livres ou ferramentas *free*, e a sigla FOSS (*Free and Open Source Software*) é usada regularmente.

3. Ferramentas de captura da interface GUI e *playback;*
4. *Frameworks* (conjunto de ferramentas que se integravam) customizáveis; e
5. *Frameworks* comerciais;

É interessante ressaltar que as ferramentas *free* para teste de *software* atualmente existentes não se encaixam em nenhuma dessas fases. Não são *frameworks* comerciais, pois pertencem à categoria de *software* livre. Algumas ferramentas são customizáveis e integráveis, outras não.

A seguir, será apresentada a metodologia adotada pela pesquisa.

4 Metodologia

O objetivo geral desta pesquisa é identificar ferramentas *free*[2] para teste de *software,* analisar e contextualizar sua atuação no processo de teste.

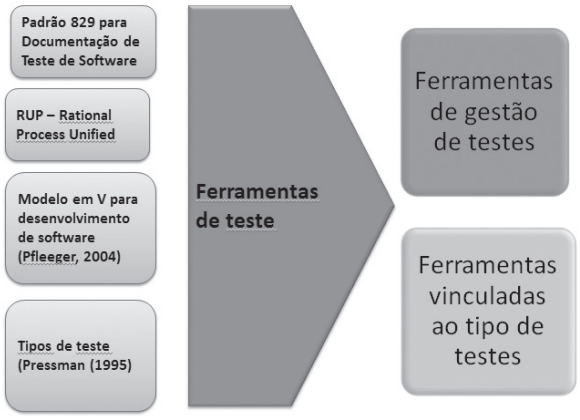
São objetivos específicos deste trabalho:

1. Conceituar o processo de teste identificando suas fases e atividades;
2. Identificar a importância do processo de teste e da automação desse processo;
3. Identificar as ferramentas *free* de teste existentes e sua atuação nesse processo; e
4. Identificar objetivos e funcionalidades de cada ferramenta.

Essa investigação utilizou a abordagem qualitativa. No que tange ao alcance temporal, a pesquisa se caracteriza como uma pesquisa interseccional, uma vez que investiga as ferramentas *free,* para teste de *software*, em determinado período, ou seja, no período da pesquisa (2011/2012). A pesquisa documental possibilitou construir o modelo conceitual adotado na pesquisa.

Nesse modelo, apresentado na Figura 5, identificam-se as várias classificações do conceito teste: relativas ao processo de teste e aos tipos de teste existentes. Assim sendo, o modelo conceitual da pesquisa propõe a classificação das ferramentas de testes em dois grupos: gestão do processo e tipos de teste. Dessa forma, modelo conceitual proposto alinha-se ao padrão IEEE 829, ao RUP, a proposta de desenvolvimento V de Pfleeger (2004), e aos tipos de teste identificados por Pressman (1995).

Figura 5 – Modelo conceitual da pesquisa



Fonte: Do autor

O método adotado é o estudo de caso, e os instrumentos de coleta de dados a serem aplicados são pesquisa documental e questionários semiestruturados para obter a percepção dos técnicos da área de testes em TI. Para análise dos dados coletados dos questionários, será utilizada a técnica de análise de conteúdo.

A utilização de variados métodos (*mixed methods research*), segundo Creswell & Clark (2007), ajuda a entender questões, que podem não ser respondidas somente pela abordagem qualitativa ou quantitativa, possibilitando a utilização de múltiplas visões de mundo.

É interessante ressaltar que o Estudo de Caso é uma investigação empírica que pesquisa fenômenos dentro do seu contexto real e no qual o pesquisador não tem controle sobre eventos e variáveis, buscando descrever, compreender e interpretar a complexidade de um caso concreto (KOLHBACHER, 2006). Algumas características são essenciais para garantir a eficácia e a credibilidade da estratégia estudo de caso, segundo Yin (2001, p.33):

- Validade do constructo – obtida através de constructos validados por revisão de literatura, utilização de múltiplas fontes, estabelecimento de caminhos de evidência. Nesse estudo fez-se uma ampla revisão de literatura com utilização de múltiplas fontes;
- Validade interna ou credibilidade – a validade e credibilidade são estabelecidas utilizan-

[2] No contexto desse trabalho, consideraram-se as ferramentas de teste tanto *free* como *open source.*

do análises e análises cruzadas, assegurando a coerência interna dos achados e utilizando a triangulação como técnica para aumento da credibilidade. Essa triangulação foi possível através do levantamento documental e da coleta das percepções dos técnicos da área de teste;

- Validade externa – garante que o escopo da pesquisa realizada pode ser replicado em casos de estudos aproximados. Envolve descrições sobre caso de estudo, protocolo de intenções dos questionários e procedimentos para codificação e análise. Os protocolos de pesquisa e instrumentos de coleta estão relacionados na pesquisa;
- Confiabilidade – Habilidade para que outros pesquisadores possam aplicar este Estudo de Caso e encontrar resultados similares. A confiabilidade foi garantida através de registro dos dados coletados para que outros pesquisadores possam seguir o caminho da evidência e replicar em outros contextos;

Considerando o público da pesquisa, o TCLE foi enviado junto com o questionário. A elaboração das questões do questionário ocorreu após a pesquisa documental, pois, os dados dessa pesquisa serviram como subsídio.

O questionário foi aplicado aos técnicos da área de TI da Embrapa, RSI e Banco do Brasil. O critério para seleção dos participantes foi definido, principalmente, com relação ao conhecimento e utilização das ferramentas de teste no ambiente organizacional, e ao tempo de trabalho na área de TI superior a 11 anos. Foi considerada também a escolaridade do respondente igual ou superior a latu senso. A escolha dessas três variáveis visa à obtenção da percepção técnica dessa solução no contexto organizacional. Com relação à idade, todos são maiores de 18 anos. Responderam ao questionário três técnicos que trabalham com testes nessas instituições.

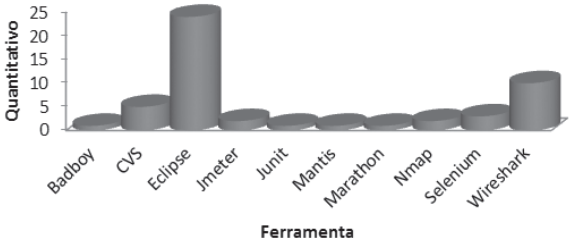
4.1 Aplicação da pesquisa e resultados

Com base nos objetivos gerais e específicos da pesquisa e para selecionar as ferramentas de teste a serem analisadas, foi definido o seguinte *string* de pesquisa que foi implementado no *google* acadêmico: ferramenta teste *software free*, ferramenta teste sistema *free*, ferramenta teste *software open source* ou ferramenta teste sistema *open source*.

Foram encontrados 359 resultados na pesquisa realizada no período de 01 ano 2011/2012. A Figura 6 apresenta o quantitativo de resultados obtidos na pesquisa efetuada. O material foi analisado e foram descartados artigos que não se relacionavam aos objetivos da pesquisa. Para selecionar o conjunto inicial de ferramentas para o estudo de todos os artigos classificados como pertinentes, foram obtidos e lidos títulos, palavras-chave e resumos, visando identificar as ferramentas citadas, funcionalidades e a sua utilização.

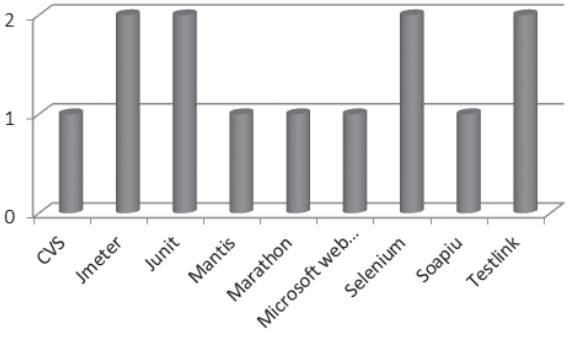
A consulta feita em livros (citados nas Referências) foi realizada observando-se os mesmos termos. A Figura 7 apresenta o quantitativo de resultados obtidos na pesquisa realizada na literatura pesquisada sobre teste.

Figura 6 - Resultado da pesquisa utilizando o string adotado



Fonte: Do autor

Figura 7 - Resultado da pesquisa utilizando o string adotado na literatura consultada



Fonte: Do autor

As ferramentas identificadas em ambas as coletas, foram classificadas em dois grupos: ferramentas de gestão do processo de teste e ferramentas vinculadas ao tipo de teste a ser realizado. Essas ferramentas são descritas a seguir.

É interessante ressaltar que, por serem ferramentas livres, foram consultados também os sites das respectivas ferramentas para obter as suas características e aspectos positivos e negativos.

5 Ferramentas de automação de teste

5.1 Ferramentas de gestão do processo de teste

5.1.1 Testlink

Segundo Molinari (2010), *Testlink* é uma ferramenta que apoia "projetos", sendo sua estrutura básica construída a partir da definição de um projeto; tudo debaixo dele será informado (requisitos de testes, especificações gerais, planos, testes, execuções etc.).

O *TestLink* é uma aplicação *Open Source* que possibilita associar um conjunto de *Test Cases* a um testador e acompanhar os resultados da execução dos testes, assim como, gerar relatórios com diversas métricas para o acompanhamento da execução dos testes. Além disso, o *TestLink* oferece um recurso para registrar e organizar os requisitos do projeto, assim como, associar os *Test Cases* aos requisitos. Dessa forma, pode-se garantir o rastreamento entre os requisitos e os *Test Cases* por meio de uma matriz de rastreabilidade (CAETANO, 2007, p. 28).

Dentre suas principais funcionalidades pode-se destacar:

- Pode ser executado em qualquer plataforma que suportar PHP/Apache/Mysql (Windows, Linux, Mac, Solaris, AS400/i5 etc.);
- Traduzido em várias línguas diferentes (incluindo "Brazil Portuguese");
- Controle de acesso e níveis de permissões por papéis (Líder, Testador, etc.);
- Criação ilimitada de projetos e casos de testes;
- Integração com ferramentas de gestão de defeitos (Bugzilla, Mantis).

5.1.2 CVS

O CVS (CVS, 2012) ou *Concurrent Version System* (Sistema de Versões Concorrentes) é uma ferramenta *Open Source* que atende os pré-requisitos básicos de um processo de Gerência de Configuração de *Software* e implementa as principais funções pertinentes ao controle de versões.

Basicamente, o CVS armazena em seu repositório as modificações realizadas num arquivo ao longo do tempo; cada modificação é identificada por um número chamado Revisão. É especialmente útil para se controlar versões de um *software* durante seu desenvolvimento, ou para composição colaborativa de um documento, pois toda Revisão no CVS armazena as modificações realizadas, quem realizou as modificações e quando foram realizadas, entre outras informações.

Além disso, o CVS conta com um mecanismo capaz de controlar os acessos simultâneos e as modificações paralelas, garantindo a integridade das modificações e a atomicidade das operações. São algumas vantagens do CVS, segundo Caetano (2007), a possibilidade de comparar diferentes versões de um arquivo, pedir um histórico completo das alterações, ou baixar uma determinada versão do projeto, ou de uma data específica, não necessariamente a versão mais atual.

Caetano (2007) cita algumas desvantagens da ferramenta, tais como: os arquivos em um repositório CVS não podem ser renomeados a partir do cliente, eles devem ser explicitamente removidos e readicionados. Não é permitido também que os diretórios sejam movidos ou renomeados. Cada subdiretório em questão deve ser individualmente removido e readicionado. Além disso, não há *checkout* reservado (permite que dois usuários alterem o mesmo arquivo, ao mesmo tempo), podendo ser mais custoso resolver o conflito do que evitar que ele ocorra.

5.1.3 Mantis

O Mantis é uma ferramenta *free*, escrita em PHP, tem seu código aberto, o que permite ao usuário a possibilidade de uma grande variedade de customizações que são incentivadas no arquivo principal do sistema. Um grande diferencial de seus concorrentes é a possibilidade de utilização do sistema em diversos idiomas, incluindo o português (CAETANO, 2007).

São 6 níveis de usuários (Visualizador, Relator, Atualizador, Desenvolvedor, Gerente e Administrador) e cada um possui diferentes níveis de atribuições e permissões dentro do sistema; um visualizador, por exemplo, pode apenas ver os registros de erros efetuados, e o administrador tem poderes completos.

Os casos reportados são separados por projetos e subprojetos; dessa forma, ficam bem organizados dentro do sistema. Assim que são salvos, os casos são designados a um usuário, que receberá um *e-mail* notificando-o como responsável pela resolução do *bug*. Após solucionar o erro, o responsável terá que responder ao remetente que avaliará se o caso pode ser encerrado ou não. Todas as ações são gravadas em *logs* que irão gerar relatórios con-

sistentes sobre o andamento dos casos dentro de um projeto (MANTIS, 2012).

Com o Mantis, um gerente de projetos, por exemplo, poderá ter um controle total dos erros encontrados pelos testadores e o tempo de resposta por parte dos desenvolvedores e quais desenvolvedores estão com quais erros. Existe também uma opção que permite a um usuário acompanhar um caso e ser avisado sobre qualquer alteração no seu registro (MANTIS, 2012).

5.1.4 Bugzilla

A ferramenta *Bugzilla* é um "Sistema de Acompanhamento de defeito" ou "*bug tracking system*". A ferramenta possibilita que indivíduos ou grupos de desenvolvedores mantenham o controle de *bugs* em seus produtos. Entre os muitos benefícios desse programa é que ele é muito escalável, mantendo os dados confidenciais e seguros, além de ter recursos avançados de pesquisa, incluindo a capacidade de armazenar as pesquisas anteriores (*BUGZILLA*, 2012).

O *Bugzilla* permite registrar um ciclo de vida para cada alteração de código efetuada no projeto, desde a sua solicitação até a sua conclusão. Com a ferramenta, é possível consultar em tempo real a quantidade de alterações do sistema, quais *bugs* estão sendo corrigidos, quem está corrigindo determinado *bug* etc. Também é considerada uma ferramenta para teste unitário.

Além disso, o *Bugzilla* oferece outras funcionalidades, tais como: gerenciamento da garantia de qualidade (QA), gerenciamento de *bugs*, fácil comunicação com os participantes do processo, segurança e confidencialidade, consultas avançadas e sistemas de permissões com perfis editáveis (*BUGZILLA*, 2012).

5.2 Ferramentas classificadas por tipos de testes

5.2.1 Badboy

BadBoy (*BADBOY*, 2012) é uma ferramenta desenvolvida em C++, não estando disponível para Linux, que grava todas as ações em uma página *web* (java, php, ruby, etc.). A ferramenta é capaz de gravar como um macro tudo que é feito na página *web* como *requests,* parâmetros, *alert*, respostas etc. Com essa ferramenta, pode-se também alterar parâmetros das páginas que estão sendo testadas, efetuar asserções por texto (simples ou html), cor, *javascript* , etc.

Algumas características da ferramenta são (*BADBOY*, 2012):

- Modo de gravação em *Internet Explorer* e *Firefox*;
- É uma ferramenta de teste funcional e teste de carga; e
- Tem capacidade de exportar para JMeter;

5.2.2 JUnit

O *JUnit* é uma ferramenta de testes unitários que fornece uma série de classes que permitem chamar outras classes (que são as que devem ser testadas) "[...] é uma ferramenta de utilização simples, além de ser *Open source* o que permite que o código seja baixado e modificado". (MOLINARI, 2010, p 65).

Em termos de funcionamento, o *JUnit* é um programa usado para executar testes unitários em virtualmente qualquer aplicação. Os testes com *JUnit* são efetuados escrevendo "casos de teste" em *Java*, compilando-os e executando as classes resultantes com o "*JUnit Test Runner*". A ferramenta permite a criação rápida de código de teste enquanto possibilita um aumento na qualidade do sistema desenvolvido e testado. Além disso, é uma ferramenta amplamente utilizada pelos desenvolvedores da comunidade código-aberto e consequentemente disponibiliza um grande número de exemplos. É interessante ressaltar que os testes, uma vez escritos, são executados rapidamente sem que, para isso, seja interrompido o processo de desenvolvimento (*JUNIT*, 2012).

Além disso, o *JUnit* possibilita checar os resultados dos testes, fornecendo uma resposta imediata. Possibilita também criar uma hierarquia de testes que permitirá testar apenas uma parte do sistema ou todo ele. Ou seja, escrever testes com *JUnit* permite ao programador a otimização do tempo gasto na depuração do código.

5.2.3 Selenium

Conforme Molinari (2010), o *Selenium* é uma ferramenta de automação de testes funcionais *web*, sendo que ela compõe um pacote maior denominado "*Open QA Selenium*", sendo uma grande concorrente com outras ferramentas pagas que dominam o mercado de testes.

O *Selenium* é uma ferramenta *Open Source* usada para a criação de testes de regressão automatizados para aplicações *WEB*. O *Selenium* foi escrito utilizando *Java*

Script e *DHTML*. Em função disso, os testes rodam diretamente a partir do navegador. Na realidade, em virtude dessa característica do *Selenium*, os testes podem rodar virtualmente em qualquer navegador que suporte *Java Script (Internet Explorer, Firefox, Opera, Safari, Konqueror* etc).

É dividido em três modos diferentes (*SELENIUM*, 2012):

- **Selenium IDE:** é uma ferramenta criada com um *plug-in* do navegador *Mozilla Firefox*, permite gravar o que é feito nesse *broswer* em um *script* cujo conteúdo tem forma de tabela e assim repetir, se necessário, as ações gravadas no *browser*.
- **Selenium Remote Control (RC):** permite salvar o que está gravado em outras linguagens, tais como *Ruby* e *Java*. Nele você pode alterar o que foi gravado acrescentando *loops* funções e bibliotecas que venham a criar.
- **Selenium Grid:** permite que você distribua seus testes em diversos computadores e os execute de forma coordenada.

5.2.4 Marathon

O *Marathon* é uma ferramenta *Open Source* de testes de regressão automatizados, para aplicações *Java* desenvolvidas com o *toolkit* gráfico *Swing*. Com essa ferramenta, podem ser criados testes automatizados por meio da captura das ações (cliques do *mouse*, digitação, etc.). Essas ações são convertidas em *scripts Jython* (implementação *Java* da linguagem de *script Python*) para que se possa executar posteriormente o teste (MARATHON, 2012).

Dentre as suas principais características, podemos destacar as seguintes (CAETANO, 2007, p.83):

- Ambiente de desenvolvimento integrado onde se pode capturar depurar e executar os testes automatizados;
- Módulos reutilizáveis para facilitar a manutenção;
- *Fixtures* (*scripts* em *Jython* cuja principal função é criar as pré-condições de execução dos testes, assim como, a posterior limpeza dos recursos criados);
- *Component* Resolver e *Custom Component* Resolver (recurso utilizado para detectar todos os componentes do *Swing* (botões, campos de edição, menus, etc.), assim como os seus métodos e propriedades).

5.2.5 Soapui

Segundo Caetano (2007), o *SoapUI* é uma ferramenta *Open Source* multiplataforma escrita em *Java* cuja principal função é testar *WEB Services*. *WEB Service* é uma tecnologia baseada em *XML* e *HTTP* cuja principal função é disponibilizar serviços interativos na *WEB* que podem ser acessados (ou consumidos) por qualquer outra aplicação independente da linguagem ou plataforma em que a aplicação foi construída.

O *soapUI* permite rapidamente criar e executar de forma automatizada testes de regressão funcional, conformidade e testes de carga. Em um ambiente de teste único, *soapUI* fornece cobertura de teste completo e suporta todos os protocolos e tecnologias. O *SoapUI* facilita todo o processo de criação e depuração dos testes por meio de uma interface gráfica intuitiva. Dentre as suas principais características, podemos destacar as seguintes (SOAPUI, 2012):

- Importação e geração automática das requisições descritas no *WSDL*;
- Capacidade de gerenciar um número ilimitado de requisições para cada operação;
- Gerenciamento de múltiplos *endpoints* para cada *WEB Service*;
- Validação das requisições e respostas contra as suas definições no *WSDL;*
- Testes funcionais, desempenho e *stress*;
- Execução de diversos testes em paralelo;
- Editores com *syntax highlight* e formatação automática;
- Suporta expressões *XPATH*; e
- Suporta criação de testes complexos utilizando *scripts Groovy*;

5.2.6 Jmeter

Apache JMeter pode ser usado para testar o desempenho tanto em recursos estáticos e dinâmicos (arquivos, *Servlets, scripts Perl*, objetos *Java,* Bases de Dados e Consultas, servidores FTP e muito mais). Ele pode ser usado para simular uma carga pesada em um servidor de rede, ou objeto para testar a resistência ou para analisar o desempenho global de diferentes tipos de carga. Pode-se realizar também uma análise gráfica de desempenho ou comportamento do objeto sob carga pesada concorrente (*JMETER,*2012).

É uma ferramenta *open source* mantida como parte do projeto Jakarta da *Apache Software Foudation* (JMETER, 2012), que fornece suporte organizacional, legal e financeiro para uma ampla gama de projetos de *software open source* (MOLINARI, 2010).

Simplificando, o *Jmeter* permite fazer testes de cargas e performance em sistemas *web*,conexões *FTP* (*File Transfer Protocol*/ Protocolo de Transferência de Arquivos), *web services*, objetos *Java* e no próprio *JUnit* testes, ou seja, testes de carga em testes de unidade que utilizam o *framework* do *JUnit*.

Principais recursos oferecidos pelo *JMeter*, segundo Caetano (2007, p. 110):

- Pode ser executado em qualquer plataforma que suporte a máquina virtual do Java 1.4 ou superior (*Solaris, Linux, Windows, OpenVMS Alpha* 7.3+);
- Suporta a criação de testes de performance para os protocolos (*HTTP, JDBC, FTP, JMS, LDAP, SOAP*, entre outros.);
- Possibilidade de executar os testes em computadores distribuídos;
- Criação de asserções para validar os requisitos de performance e funcionalidade;
- Possibilidade de monitorar a performance de um servidor *Apache Tomcat;*
- Permite a utilização de pré-processadores e pós-processadores para modificar o comportamento das requisições; e
- Suporta diversos tipos de monitores para avaliar a performance da aplicação em teste;

5.2.7 Microsoft web application stress

Segundo afirma Caetano (2007), *Microsoft WEB Application Stress* (WAS) é uma ferramenta gratuita para o *Windows*. Por meio do *WAS*, você poderá realizar testes de performance, volume e estresse nas suas aplicações *WEB*.

O *WAS* é uma ferramenta que possui menos recursos que o *JMeter*, no entanto, ela possibilita testes rápidos. Os testes podem ser escritos manualmente, por meio da criação manual das requisições *HTTP* (GET, POST, etc.), ou gravados automaticamente enquanto você navega na aplicação simulando um usuário real por meio de um *Proxy Server* (CAETANO, 2007).

Entre os diversos recursos oferecidos pelo *WAS*, deve-se destacar a possibilidade de executar os testes em computadores distribuídos, a configuração do comportamento do teste de performance e a possibilidade de monitorar a utilização dos recursos dos servidores *Windows* (apenas no *Windows* 2000 e NT).

5.2.8 Wireshark

O *Wireshark*, anteriormente conhecido como *Ethereal*, é um programa que analisa o tráfego de rede e o organiza por protocolos. O primeiro passo na avaliação de vulnerabilidades é ter uma imagem clara do que está acontecendo na rede. O *Wireshark* funciona em modo promíscuo para capturar todo o tráfego de um domínio de *broadcast TCP*. (WIRESHARK, 2012).

Filtros personalizados podem ser ajustados para interceptar o tráfego específico, por exemplo, para capturar a comunicação entre dois endereços IP, ou capturar consultas DNS baseados na rede. Os dados de tráfego podem ser despejados em um arquivo de captura, que pode ser revisto mais tarde. Os filtros adicionais também podem ser definidos durante a revisão.

Normalmente, o testador está à procura de endereços IP vadios, pacotes com endereços forjados, pacotes desnecessários e geração de pacotes suspeitos a partir de um único endereço IP. O *Wireshark* dá uma visão ampla e clara do que está acontecendo na sua rede.

5.2.9 Nmap

Segundo *Nmap* (2012), a ferramenta *Nmap* (*Network Mapper*), é considerada como código aberto para exploração de rede e auditoria de segurança. Ela foi desenhada para escanear rapidamente redes amplas, embora também funcione muito bem em *hosts* individuais. O *Nmap* utiliza pacotes IP em estado bruto (*raw*) para determinar quais *hosts* estão disponíveis na rede, quais serviços (nome da aplicação e versão) os hosts oferecem, quais sistemas operacionais (e versões de sistemas operacionais) eles estão executando, que tipos de filtro de pacotes/*firewalls* estão em uso, e dezenas de outras características.

Embora o *Nmap* seja normalmente utilizado para auditorias de segurança, é útil para tarefas rotineiras tais como inventário de rede, gerenciamento de serviços de atualização agendados, monitoramento de *host* ou disponibilidade de serviço.

A saída do *Nmap* é uma lista de alvos escaneados, com informações adicionais de cada um dependendo das opções utilizadas. Uma informação chave é a "tabela de portas interessantes". Essa tabela lista o número

da porta e o protocolo, o nome do serviço e o estado (*NMAP*, 2012).

O estado pode ser aberto (*open*), filtrado (*filtered*), fechado (*closed*), ou não-filtrado (*unfiltered*). Aberto (*open*) significa que uma aplicação na máquina-alvo está escutando as conexões/pacotes naquela porta. Filtrado (*filtered*) significa que o *firewall*, filtro ou outro obstáculo de rede está bloqueando a porta de forma que o *Nmap* não consegue dizer se ela está aberta (*open*) ou fechada (*closed*). Portas fechadas (*closed*) não possuem uma aplicação escutando nelas, embora possam abrir a qualquer instante.

Portas são classificadas como não filtradas (*unfiltered*) quando elas respondem às sondagens do *Nmap.* Mas o *Nmap* não consegue determinar se as portas estão abertas ou fechadas. O *Nmap* reporta as combinações aberta/filtrada (*open/filtered*) e fechada/filtrada (*closed/filtered*), quando não consegue determinar qual dos dois estados descreve melhor a porta. A tabela de portas também pode incluir detalhes de versão de *software* quando a detecção de versão for solicitada. Quando um *scan* do protocolo IP é solicitado (-sO), o *Nmap* fornece informações dos protocolos IP suportados ao invés de portas que estejam abertas.

Utiliza recursos avançados para verificar o estado do seu alvo. A ferramenta é gratuita e encontrada nas versões *Linux, Windows* (95, 98, NT, Me, 2K e XP), *Mac* OS, *Solaris*, *FreeBSD* e *OpenBSD*. As versões para *Windows* e *Linux* contam com uma interface gráfica que facilitam a vida dos usuários.

O *Nmap* detecta dispositivos remotos e, na maioria dos casos, identifica *firewalls* e roteadores, além de sua marca e modelo. Pode-se usar o *Nmap* para verificar quais portas estão abertas, e também se essas portas podem ser exploradas ainda mais em ataques simulados.

5.2.10 Eclipse TPTP

O Teste de Eclipse (ECLIPSE, 2012) e Performance *Tools Platform Project* (TPTP), fornece uma plataforma aberta e serviços que permitem aos desenvolvedores de *software* construírem teste único e ferramentas de desempenho, tanto *open source* e comercial, que podem ser integrados com plataformas e com outras ferramentas.

O Eclipse não é uma IDE propriamente dita; é um arcabouço para desenvolvimento de ferramentas, sendo extensível, aberto e portável. Por meio de *plugins*, diversas ferramentas podem ser combinadas, criando um ambiente de desenvolvimento integrado. A ferramenta possibilita assim, uma mesma plataforma ser utilizada para vários papéis de desenvolvimento: programador de componentes, integrador, responsável por testes, *web designer*.

A ferramenta engloba todo o teste e ciclo de vida, desempenho de testes iniciais para monitoramento de aplicações de produção, incluindo a edição de teste e execução, monitoramento, rastreamento e perfis, e capacidades de análise de *log*. A plataforma suporta um amplo espectro de sistemas de computação de alto desempenho, múltiplos ambientes *Java*, substituição dinâmica de código, refatoração etc.

O Eclipse possui, ainda, modelos de ajuda prontos (*Wizards*) para efetuar várias tarefas. Criar testes usando *JUnit* é uma delas.

6 Correlação entre modelos vs tipos de teste vs ferramentas

A Tabela 1 apresenta as ferramentas de teste analisadas e classificadas em dois grandes grupos: Ferramentas de gestão do processo de teste e Ferramentas vinculadas ao tipo de teste. Além disso, essa tabela identifica, a partir dos modelos citados, os tipos de testes e as ferramentas de testes que podem ser utilizadas para a automatização e gerenciamento desse processo. Por exemplo, a identificação da abordagem de teste a ser utilizada (*RUP*) e a especificação do projeto de teste (segundo padrão IEEE 829) poderiam englobar a utilização das ferramentas *Test Link*, CVS e *Mantis*.

Tabela 1 – Correlação modelos vs processo vs ferramentas.

Assunto	**Abordagem**				
	Processo RUP	**Padrão IEEE 829 para documentação de teste**	**Tipo de teste (Pressman, 1995)**	**Classificação de ferramentas (Caetano, 2007)**	**Ferramentas**
	-Definição da missão -Verificação da abordagem	**Plano de teste** Especificação do projeto de teste	Não se aplica	**Gestão de testes**	Test link
				Controle de versão	CVS
				Gestao de defeitos	Mantis Bugzila
	Definição da missao Verificação da abordagem	**Plano de teste** Especificação do projeto de teste	-Teste de validação -Teste de sistema	**Automação de testes funcionais e de aceitação**	Seleniun
					Marathon
					Soapui Badboy
			-Teste de desempenho -Teste de stess	**Automação de teste de performance**	Jmeter Eclipse TPTP
					Microsoft web application stress
			-Segurança	Não se aplica	Wireshark Nmap
		Especificação dos casos de teste	-Teste de unidade -Teste integração	**Gerencia de casos de teste**	Badboy Junit Bugzilla

Fonte: Do autor

A seguir, serão descritos sucintamente os principais resultados da análise de conteúdo correspondente à aplicação dos questionários e considerando as ferramentas estudadas.

7 Resultados da aplicação dos questionários

As respostas obtidas pela aplicação do questionário demonstraram, nas empresas Embrapa, RSI e Banco do Brasil, que:

- A utilização da ferramenta Mantis, em duas das organizações pesquisadas, foi avaliada como atendendo satisfatoriamente seus objetivos.
- Uma das organizações pesquisada utiliza apenas teste de sistema;
- Uma das organizações, trabalha somente com testes de unidade e desempenho.
- Uma das organizações utiliza todos os tipos de teste que são: teste de unidade, teste de integração, validação, sistema, desempenho, *stress* e segurança.
- *Testlink, CVS, Bad boy* foram avaliadas por um dos participantes como atendendo satisfatoriamente os objetivos.
- *Jmeter* foi avaliada como atendendo parcialmente os objetivos.
- *Cvs* e *Selenium* foram avaliadas por um dos participantes como atendendo satisfatoriamente os objetivos.

Um dos participantes ressaltou [3] "[...] a necessidade de no questionário estarem listadas também as ferramentas de teste para sistema legado". Importante ressaltar que o foco da pesquisa foram ferramentas *free*, e que a maior parte de ferramenta de teste para legado são ferramentas pagas.

[3] Questionário aplicado em técnicos da área de TI da Embrapa, RSI e Banco do Brasil.

8 Conclusão

O objetivo geral deste trabalho foi identificar as ferramentas *free* para teste de *software*, analisando e contextualizando sua atuação no processo de teste. Para isso, foram descritos o conceito de teste, processo de teste e identificadas as suas fases e atividades. Dessa forma atendeu-se ao objetivo específico 1.

Foi também demonstrada a importância do processo de teste e da automação deste processo. O estudo demonstra que o processo e sua automação são essenciais para o desenvolvimento e manutenção de um *software* de qualidade.

Foram selecionadas algumas das ferramentas *free* de teste, já referenciadas academicamente e em livros, e sua atuação no processo. Dentre as ferramentas estudadas, identificaram-se vários recursos e funcionalidades. É interessante ressaltar que essas ferramentas estão à disposição para qualquer usuário baixá-las a qualquer momento. Dessa forma, a pesquisa atendeu aos objetivos específicos 2 e 3.

Além disso, as ferramentas de teste foram relacionadas às fases do processo de teste. Foram pesquisadas na literatura e na *Web* as vantagens e desvantagens dessas ferramentas.

Para complementar o trabalho, foi realizado estudo de caso em três instituições visando obter a percepção dos técnicos da área de testes de TI sobre a utilização dessas ferramentas. Identificou-se que a ferramenta *Mantis* é a mais utilizada nas instituições pesquisadas e tem uma avaliação satisfatória.

Sugere-se para estudos futuros a aplicação do questionário em outras organizações visando poder comparar a percepção de mais participantes sobre as ferramentas citadas.

Referências

BADBOY. Disponível em: <www.badboy.com.au/>. Acesso em: 03 mar. 2012.

BUGZILLA. Disponível em: <http://www.bugzilla.org/>. Acesso em: 05 abr. 2012 .

CAETANO, C. *Automação e gerenciamento de teste*. Rio de Janeiro: Codeline. Comércio e tecnologia, 2007. E-book. Disponível em: <http://www.linhadecodigo.com.br>. Acesso em: 05 maio 2011.

CRESWELL, John W.; CLARK, Vicki L. Plano. *Designing and conducting mixed methods research*. California: Sage Publications, 2007.

CVS. Disponível em: <http://ximbiot.com/cvs/manual/>. Acesso em: 03 mar. 2012.

DELAMARO, E.; MALDONADO, J. C.; JINO, M. *Introdução ao teste de software*. Rio de Janeiro: Elsevier, 2007.

ECLIPSE. Disponível em: <http://www.eclipse.org/tptp/home/>. Acesso em: 03 mar. 2012.

HAYES, L.G. Evolution of automated software testing. In: AUTOMATED software testing magazine. *Automated testing institute,* v. 1, n. 2, p.14-20, ago. 2009.

IEEE . *Standard for Software & System*. Test Documentation. IEEE 829-2008. New York: IEEE, 2008.

IEEE Std 610. *Standard Glossary of Software Engineering Terminology*, IEEE Std 610. New York: IEEE,1990.

JMETER. Disponível em: <http://jmeter.apache.org/>. Acesso em: 05 abr. 2012.

JUNIT. Disponível em: <http://www.junit.org/>. Acesso em: 03 mai. 2012.

KOLHBACHER, Florian. The use of qualitative content analysis in the case study research. *Forum*: Qualitative Social Research , Berlin, v. 7, n. 1 , art. 21, jan. 2006.

MANTIS. Disponível em: <http://www.mantisbt.org/>. Acesso em: 03 maio 2012 .

MARATHON. Disponível em: <http://sourceforge.net/projects/marathonman/> Acesso em: 03 mar. 2012.

MOLINARI, Leonardo. *Inovação e automação de testes de software*. São Paulo: Erika, 2010.

MOLINARI, Leonardo. *Teste de software*. São Paulo: Erica, 2003.

NMAP. Disponível em: <http://nmap.org/man/pt_BR/index.html#man-description>. Acesso em: 03 maio 2012.

PFLEEGER, Shari L. *Engenharia de software*: teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

PONTES, Melissa Barbosa. Introdução a testes de software. *Engenharia de Software Magazine*, ano 1, 2009.

PRESSMAN, Roger S. *Engenharia de Software*. São Paulo: Makron books, 1995.

RATIONAL SOFTWARE. *Rational Unified Process 7.0.1*. IBM, 2008. Disponível em: <http://www.wthreex.com/rup/>. Acesso em: 20 jan. 2012.

SELENIUM. Disponível em: <http://seleniumhq.org/>. Acesso em: 07 jul. 2012.

SOAPUI. Disponível em: <http://www.soapui.org/>. Acesso em: 03 maio 2012.

SOMMERVILLE, Ian. *Engenharia de software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

WIRESHARK. Disponível em: <http://www.wireshark.org/>. Acesso em: 13 abr. 2012.

YIN, Robert K. *Estudo de caso:* planejamento e métodos. 2. ed. Porto Alegre: Bookman, 2001.