UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CHARLAN BETTIOL

DESENVOLVIMENTO DE UM PROTÓTIPO PARA UNIÃO DAS FERRAMENTAS DE AUTOMAÇÃO E GERÊNCIA DE TESTES DE SOFTWARE

CRICIÚMA
2015

CHARLAN BETTIOL

DESENVOLVIMENTO DE UM PROTÓTIPO PARA UNIÃO DAS FERRAMENTAS DE AUTOMAÇÃO E GERÊNCIA DE TESTES DE SOFTWARE

Trabalho de Conclusão de Curso, apresentado para obtenção de grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Esp. Gilberto Vieira da Silva

CRICIÚMA
2015

CHARLAN BETTIOL

DESENVOLVIMENTO DE UM PROTÓTIPO PARA UNIÃO DAS FERRAMENTAS DE AUTOMAÇÃO E GERÊNCIA DE TESTES DE SOFTWARE

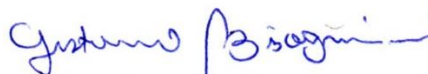
Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Qualidade de Software.

Criciúma, 26 de Novembro de 2015.

BANCA EXAMINADORA

Prof. Esp. Gilberto Vieira da Silva - UNESC - Orientador

Prof. Esp. Fabrício Giordani - UNESC



Prof. MSc. Gustavo Bisognin - UNESC

Dedico este trabalho a todos que me ajudaram a concluir esta etapa de minha caminhada.

AGRADECIMENTOS

Quero agradecer aos meus pais que me deram apoio e nunca mediram esforços para melhorar minha educação e me dar o caminho correto a seguir.

As minhas irmãs Schaiane, Chalana e Dominique que sempre me ajudaram e acompanharam em toda minha vida acadêmica.

A minha namorada, Janaina pelo carinho, compreensão e paciência demostrado em toda minha jornada como acadêmico.

Agradeço a meu primeiro coordenador na área de tecnologia Márcio Figueiredo Cardoso ao qual me deu a oportunidade para trabalhar com o desenvolvimento de softwares e sempre me incentivou para alcançar meus objetivos.

A empresa Raiss sistema por me dar a oportunidade de desenvolver me trabalho utilizando seu produto o tempo da empresa.

Agradeço ao meu orientador Prof. Esp Gilberto Vieira da Silva, ao qual abriu mão de suas horas diárias para poder me orientar e dar o caminho correto para o desenvolvimento do trabalho.

Por fim, agradeço a todos colegas e professores do curso de Ciência da Computação.

“Ou expulsamos de nós a Alma da Derrota ou nem vale a pena competir”

Nelson Rodrigues

RESUMO

O mercado de desenvolvimento de software cresce rapidamente a cada dia e com esse aumento observa-se uma exigência de qualidade cada vez maior. As empresas sabem que para garantir a qualidade dos softwares é necessário ter presente na equipe de desenvolvimento uma equipe de testes para assegurar a qualidade da solução implementada. Para certificar a qualidade as atividades de teste têm o papel fundamental de encontrar defeitos no software, diminuindo assim o trabalho do desenvolvedor em corrigi-los posteriormente com a solução já rodando em clientes. Neste contexto a execução manual de um teste é rápida e efetiva, porém a mesma realização em um conjunto de testes e a repetição deste é uma tarefa árdua e demorada. No sentido em questão a automação de testes aumenta a produtividade e atinge em um tempo menor e com a mesma eficácia os casos de testes mais maçantes e repetitivos. Para facilitar a comunicação entre estas duas equipes, já existem ferramentas especificas, os gerenciadores de *bugs* que visam facilitar o processo de desenvolvimento de um software. Neste contexto o presente trabalho propõe através do estudo da ferramenta *Open Source Mantis Bug Tracker* criar um protótipo para unir-se a ferramenta de automação de testes *Selenium* para que de forma automática os *bugs* encontrados através dos scripts de testes sejam reportados a equipe de desenvolvimento.

Palavras-chave: Qualidade de software.Teste de Software. Automação de testes. Gerenciadores de *bugs*. Ferramenta Mantis.

ABSTRACT

The software development market is growing rapidly and with this expansion it can be noticed an increasing demand for quality. The companies know that to ensure the software quality is necessary to have a testing team in the development unit to assure the quality regarding the implemented solution. In order to certify the quality the test activities have the crucial role of finding software defects and thereby decreasing the developer's job to correct them later with the solution already running in customer's environment. In this context the test manual execution is fast and effective, but the same implementation in a set of tests and its repetition is an arduous and time consuming task. In this way the testing automation increases the productivity and reaches in a shorter time and with the same effectiveness the cases concerning the repetitious and tiresome tests. To make it easier the communication between these two teams there are specific tools. These are the bug managers, that try to ease the software development process. In this scenario the present essay proposes, by using the Open Source Mantis Bug Tracker tool, to create a prototype to join up with the Selenium testing automation tool in order to automatically report the bugs found by the testing scripts.

Keywords: Software Quality. Software testing. Automation testing. Bug managers. Mantis tool.

LISTA DE ILUSTRAÇÕES

Figura 1 - Ciclo de vida clássico da engenharia de software (modelo cascata). 17
Figura 2 – Componentes MPS.BR 23
Figura 3 – Etapas do processo de teste 26
Figura 4 – Processo de teste de software 29
Figura 5 - Níveis, tipos, técnicas e critérios para definição de estratégias de teste .. 30
Figura 6 – Definição da Norma IEEE 829:2008 para o plano de trabalho. 36
Figura 7 – Bugzilla 38
Figura 8 – Mantis *Bug Tracker* 39
Figura 9 – Fluxo de erro 40
Figura 11 – Tela inicial do RaissTrans 52
Figura 12 – Tela inicial do Mantis bug tracker 53
Figura 13 – Tela de cadastro de erros do Mantis bug tracker 54
Figura 14 – serviço mantisconnect 54
Figura 15 – Ciclo de vida dos testes 55
Figura 16 – Ciclo de vida do protótipo 55
Figura 17 – Base de testes 56
Figura 18 – Script gravados no Selenium IDE 58
Figura 19 – Exportar scripts para Java 58
Figura 20 – Script exportado para extensão .java 59
Figura 21 – Script importado na IDE de desenvolvimento 60
Figura 22 – Classe de conexão com Mantis 61
Figura 23 – Classe das constantes de conexão 62
Figura 24 – atributos da *ISSUE* 62
Figura 25 – Anexar captura de tela 63
Figura 26 – *ISSUE* cadastrada no Mantis 63
Figura 27 – Configurações do arquivo no Mantis 64
Figura 28 – Configurações de e-mails no Mantis 65
Figura 29 – E-mail recebido na caixa de entrada 65
Figura 30 – E-mail aberto com todas informações 66

LISTA DE ABREVIATURAS E SIGLAS

AP	Atributos de Processos
CMMI	Capability Maturity Model Integration
GUI	Graphical User Interface
ISO/EIC	International Organization for Standardization and the international Electrotechnical Commission
MA-MPS	Método de Avaliação
MN-MPS	Modelo de Negócio
MPS.BR	Melhoria do Processo do Software Brasileiro
MR-MPS	Modelo de Referência
PSEE	Process Centered Software Engineering Environment
RAP	Resultados esperados Atributos do Processo
SEI	Software Engineering Institute

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVO GERAL 12
1.2 OBJETIVOS ESPECÍFICOS 13
1.3 JUSTIFICATIVA 13
1.4 ESTRUTURA DO TRABALHO 14
2 ENGENHARIA DE SOFTWARE 16
2.1 QUALIDADE DE SOFTWARE 18
2.2 GARANTIA DA QUALIDADE DE SOFTWARE 21
2.3 MODELO DE QUALIDADE MPS.BR 22
3 TESTE DE SOFTWARE 25
3.1 MODELO “V” DE TESTE DE SOFTWARE 28
3.2 TÉCNICAS DE TESTE DE SOFTWARE 29
3.2.1 Teste caixa branca 30
3.2.2 Teste caixa preta 31
3.2.3 Teste unitário 34
3.2.4 Teste funcional 34
3.3 GERÊNCIA DE TESTES 35
3.3.1 Bugzilla 38
3.3.2 Mantis *bug tracker* 39
3.3.3 TestLink 41
3.3.4 Análise da aderência das ferramentas no contexto da criação do protótipo 42
4 TESTES AUTOMATIZADOS 43
4.1 FERRAMENTAS DE TESTE 44
4.2 MÉTODOS DE AUTOMAÇÃO DE TESTES 45
5 TRABALHOS CORRELATOS 47
5.1 DESENVOLVIMENTO DE UMA MÉTODOLOGIA APLICADA AO GERENCIAMENTO E AOMPANHAMENTO DE TESTES DE SOFTWARE VIA WEB 47
5.2 PROCESSO DE AUTOMAÇÃO DE TESTES DE SOFTWARE COM FERRAMENTA OPEN SOURCE: UM ESTUDO DE CASO COM INTEGRAÇÃO CONTINUA 48

5.3 FRAMEWORK FUNCTEST: APLICANDO PADRÕES DE SOFTWARE NA AUTOMAÇÃO DE TESTES FUNCIONAIS 49
5.4 DEF-PRO: APOIO AUTOMATIZADO PARA A DEFINIÇÃO DE PROCESSOS DE SOFTWARE 49
5.5 DESENVOLVIMENTO DE UMA METODOLOGIA BASEADA NA AUTOMAÇÃO O PROCESSO DE TESTES DE SOFTWARE COMO ESTRATÉGIA PARA A GARANTIA DA COBERTURA FUNCIONAL 50
6 DESENVOLVIMENTO DO PROTÓTIPO PARA AUTOMATIZAR TESTES E REPORTAR DEFEITOS AUTOMATICAMENTE 51
6.1 METODOLOGIA 51
6.2 SOLUÇÃO A SER TESTADA 52
6.4 FERRAMENTA DE GERÊNCIA DE TESTES UTILIZADA 53
6.5 DEFINIÇÃO DO CICLO DE VIDA DO PROTÓTIPO 54
6.6 BASE DE TESTES 55
6.7 TESTES AUTOMATIZADOS 57
6.8 REPORTAR ERROS 61
6.9 NOTIFICANDO O DESENVOLVEDOR 64
7 CONCLUSÃO 67
REFERÊNCIAS 68
ANEXO(S) 71
APÊNDICE A – ARTIGO CIENTÍFICO 73

1 INTRODUÇÃO

Atualmente o mercado tecnológico apresenta uma grande demanda de desenvolvimento de softwares, juntamente com esta exigência computacional a busca pela qualidade de software aumenta, em contrapartida os prazos curtos e escopos complexos acabam gerando uma baixa qualidade computacional dos sistemas desenvolvidos. A alta solicitação e necessidade de entregas em curto tempo, gera para as entidades desenvolvedoras a necessidade de buscar um meio para que a entrega seja feita no prazo e com a qualidade desejável pelo cliente final.

As entidades desenvolvedoras junto dos avanços tecnológicos buscam melhores métodos para desenvolver produtos e serviços, diminuindo o tempo, esforço e custos aplicados. Por outro lado, a tecnologia também acrescenta desafios mais complexos ao requerer uma integração do software a sistemas e componentes de terceiros (CHRISSIS; KONRAD; SHRUM, 2007, tradução nossa).

Conforme Pressman (2006), o software possui uma linha evolutiva e apresentará defeitos durante a existência, alguns devido à evolução, pois quanto maior a complexidade, aliada a manutenções constantes, maiores serão as falhas e a queda na qualidade de software.

Segundo Prikladnicki (2006) o desenvolvimento de sistemas de software parece estar relacionado a uma técnica ortodoxia, sendo vista apenas como um processo totalmente técnico a ser executado especificamente por especialistas. O esforço necessário para a produção de sistemas deste tipo apresenta problemas e desafios de complexidade muito além da técnica. Ou seja, necessitam de conhecimentos avançados provenientes de aspectos multidisciplinares, oriundos de outras áreas do conhecimento.

Algumas organizações defendem que a implantação de testes de software serve apenas para redução de custos de problemas. Líderes de equipes, gerentes de testes, arquitetos de software e analistas de testes defendem que investir em qualidade de software é importante para que a aplicação seja conforme as necessidades e expectativa criada sobre o produto desenvolvido. Dessa forma garante-se que o software desenvolvido estará de acordo com os requisito e especificações mínimas exigidas. Como consequência, a redução de custos de manutenção, satisfação do usuário e redução de incertezas que cercam o software

são garantidas.

Automatizar testes significa reproduzir os passos da interação de um software de forma automática através de outra aplicação tal como um usuário estivesse reproduzindo os passos. A necessidade de realizar verificações em menor tempo e com a mesma qualidade justifica a disseminação da automação de testes de software, conforme as aplicações ficam mais complexas ao mesmo tempo se passou a exigir maior qualidade de produtos entregues (MOLINARI, 2010).

Visando minimizar os custos e tempo de produção e com a garantia de qualidade final do produto entregue, o uso da automação de testes requer um gerenciamento de testes de software, tendo em vista que é um investimento de longo prazo, que engloba reestruturação e reorganização de equipes e processos, e em alguns casos a aquisição de ferramentas, a produção de alguns impactos também precisa ser avaliada de forma aproximada, como o aumento de tempo para reportar defeitos encontrados e manutenções de scripts e preparações dos testes (RIOS; MOREIRA, 2006).

Diante dos pontos apresentados o presente trabalho tem por objetivo criar um protótipo para automatizar testes de software e reportar automaticamente as falhas encontradas através da união de ferramentas de gerência de testes e automação de testes, o protótipo deverá garantir que os erros encontrados sejam reportados de forma automática sem a necessidade de interação do analista de testes, garantindo assim a qualidade mínima exigida pelos requisitos e a redução de tempo e custo associados ao desenvolvimento do projeto.

1.1 OBJETIVO GERAL

Desenvolver um protótipo, focado no processo automação e gerenciamento de testes de software para reportar automaticamente os defeitos encontrados ao desenvolvedor, garantindo a cobertura funcional e a qualidade durante o processo de validação registrando os defeitos na ferramenta de gerência de testes.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste Trabalho de Conclusão de Curso (TCC) são:

a) estudar a área de qualidade de software;

b) analisar ferramentas de automação de testes;

c) analisar ferramentas de gerencia de testes;

d) definir ferramenta de automação e gerencia de testes;

e) desenvolver um protótipo para validar as metodologias abordadas.

1.3 JUSTIFICATIVA

As falhas no processo de desenvolvimento de um software são causadas por má implementação ou por falha na especificação de uma funcionalidade, todo o tipo de sistema instável causa prejuízos como, atrasos em atividades, retrabalho, perda de informações dentre outros.

Segundo Avizieniset al (2004), um defeito ou uma falha é um acontecimento que ocorre quando o software desenvolvido não fornece as suas funções conforme a especificação do cliente.

Para Yourdon (1990), nenhum sistema computacional está livre de falhas. Os erros ou falhas podem permanecer em um software por toda sua vida, devido à quantidade de funcionalidades pouco ou nunca utilizadas.

Levando em questão a complexidade de tamanho dos softwares desenvolvidos a cobertura de testes funcional e a automática faz com que uma quantidade maior de defeitos seja identificada ainda na fase de desenvolvimento, ficando claro que é muito difícil atingir 100% da cobertura de todos os requisitos do software. Com isso, as técnicas de testes podem ser aplicadas por meio de ferramentas que permitem a realização dos testes de forma não assistida, aumentado a capacidade de cobertura de testes funcionais em um menor período de desenvolvimento (CAETANO, 2008).

Na atualidade existem diversas ferramentas para aplicação de automação de testes de software. Sendo por meio de repetição de ação de usuários (*Capture-Replay*) ou por meio de desenvolvimento de scripts de testes (*data-driven*), estas ferramentas permitem a execução e a validação dos resultados esperados.

Juntamente com a execução dos scripts de testes automatizados surgem erros de desenvolvimento ou de regra de negócio e com estes surge a necessidade de reportar os mesmos. Essa é suprida pelo meio de integração de ferramentas de execução de scripts de testes automatizados e ferramentas de gerência de testes, onde as ferramentas de gerência de testes fornecem diferentes meios para se reportarem os defeitos como relatórios, banco de dados, e-mail entre outros. A união dos testes automatizados com as ferramentas de gerência de testes torna o desenvolvimento de um software mais ágil e rápido, sendo assim o testador não precisar executar os testes automatizados de forma assistida, onde essa ação pode gerar diminuição de custos e tempo hábil a empresa fornecedora de software (KELLING, 2008 apud SAMPAIO, 2012).

A metodologia proposta neste trabalho visa explorar as ferramentas de gerência de testes como o Mantis *BugTracker* para a organização do projeto de testes, e juntamente com esta analisar as ferramentas de testes automatizados do tipo *data-driven* como a Selenium Driver para apoiar a execução de testes funcionais automatizados na produção de softwares, deste modo o desenvolvimento do protótipo com enfoque na união das ferramentas Mantis *BugTracker* e Selenium Driver para que possa-se criar testes automatizados não assistidos e um meio para que sejam reportados automaticamente os defeitos encontrados no processo de desenvolvimento a partir da execução de scripts de testes.

O principal motivo que se teve em consideração a realização da pesquisa e desenvolvimento do protótipo foi o estabelecimento de padrões de gerencia de testes e criação de scripts de testes automatizados reportando os erros de forma automática, propiciando assim a otimização de tempo no processo de testes e uma maior cobertura funcional de um software reduzindo os defeitos causados por falhas no processo dos mesmos.

1.4 ESTRUTURA DO TRABALHO

Este trabalho divide-se em uma estrutura de seis capítulos.

O primeiro trata da introdução referente ao projeto expondo os objetivos gerais e específicos e finalizando com a justificativa para realização do trabalho.

No segundo é contextualizado sobre a engenharia de software fomenta

sobre os padrões para o processo de desenvolvimento de um software focando na área de qualidade.

No terceiro capitulo apresenta-se os processos e detalhamentos que envolvem os testes de software, abortando os tipos, métodos e ferramentas para gerenciar os mesmos.

Apresentando a parte do estudo automatização de testes o quarto capitulo traz juntamente com os métodos de automação de testes as ferramentas para que a mesma aconteça, apresentando as ferramentas TestComplete e Selenium.

O *brainstorm* para criação desse trabalho foi possível através das pesquisas de trabalhos correlatos que se encontram apresentados no capítulo cinco.

Finalmente apresentado pelo capitulo seis a metodologia e a criação do protótipo proposto.

2 ENGENHARIA DE SOFTWARE

A Engenharia de Software trata-se de uma área do conhecimento da informática voltada para a especificação, desenvolvimento e manutenção dos sistemas de software aplicando tecnologias e práticas de gerência de projetos, objetivando organização, produtividade e qualidade (PRESSMAN, 2006).

A engenharia de software abrange um conjunto de três elementos principais, sendo eles: métodos, ferramentas e procedimentos. A união desses elementos é o que dá possibilidade ao gerente de software o controle do processo de desenvolvimento oferecendo assim aos profissionais envolvidos uma base para a construção de um software de alta qualidade e produtividade (PRESSMAN, 1995).

Segundo Prikladnicki (2006) o desenvolvimento de sistemas de software está relacionado a uma técnica ortodoxia, sendo visto apenas como um processo totalmente técnico a ser executado especificamente por especialistas. O esforço necessário para a produção de sistemas deste tipo apresenta problemas e desafios de complexidade muito além da técnica. Ou seja, necessitam de conhecimentos avançados provenientes de aspectos multidisciplinares, oriundos de outras áreas do conhecimento.

Essas visões e características convergem para a definição de que o software é um produto imaterial e flexível, elaborado em um crítico processo produtivo, que une a visão e conhecimento técnico do desenvolvimento às necessidades funcionais objetivadas pelo cliente final. Segundo Burnstein (2010, tradução nossa) e Sommerville (2003), no âmbito da qualidade. O software ainda precisa comportar quatro atributos básicos, que são a manutenabilidade, confiabilidade, eficiência e praticidade.

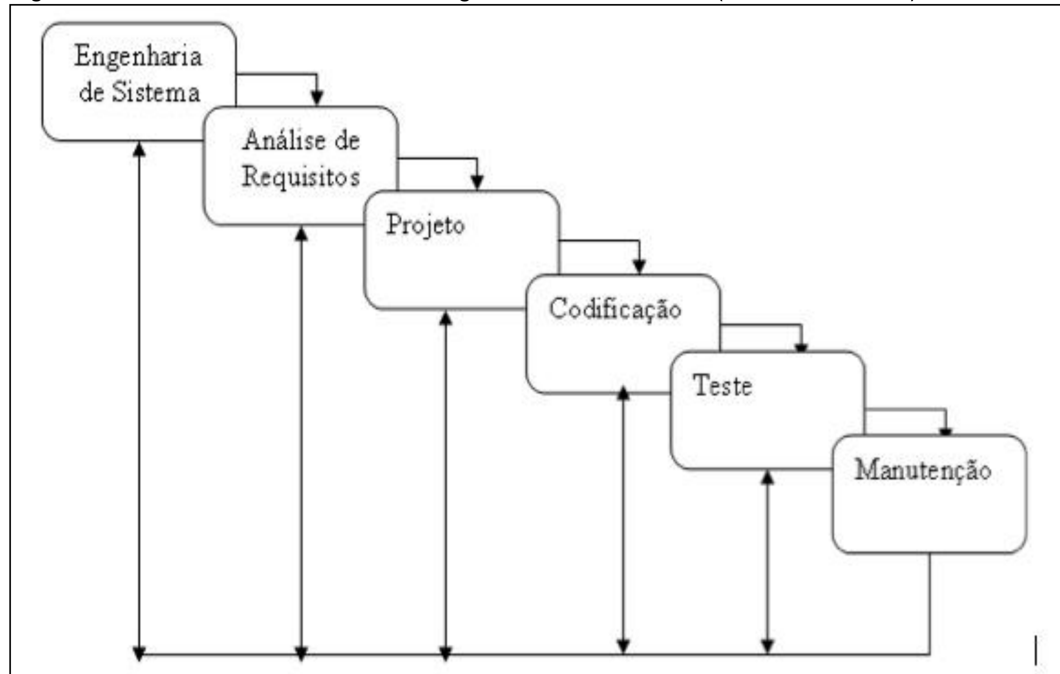
A construção de softwares deve obedecer estes aspectos deste modo seguir as características de qualidade usabilidade, modularização, robustez e confiabilidade. Devido as constantes alterações nos projetos de desenvolvimento de software tais características tornam-se difíceis de serem mantidas.

Partindo dessa ótica, a abordagem de engenharia aplicada à produção de software, tem a importante missão de agregar qualidade, produtividade e redução de custos (PETERS; PEDRYCZ, 2000, tradução nossa; SOMMERVILLE, 2003).

A figura 1 ilustra um dos ciclos de vida mais comum utilizado no processo

de desenvolvimento de software, conhecido como modelo cascata, este demonstra uma forma sequencial do ciclo de desenvolvimento de um software.

Figura 1 - Ciclo de vida clássico da engenharia de software (modelo cascata).



Fonte: Adaptado de Pressman (1995).

Além do modelo cascata, existem outros modelos de desenvolvimento, onde para cada ambiente deve ser utilizado o modelo que melhor se encaixa.

No ciclo de vida cascata as fases do projeto são executadas em uma sequência ordenada, onde a próxima só poderá ser iniciada quando houver o termino da etapa anterior.

No ciclo de vida espiral o projeto é separado em pequenos pedaços, tornando o projeto inicial constituído em pequenos projetos, deste modo é possível eliminar os problemas de grandes proporções por meio de projetos gerenciados e de um planejamento estruturado.

Além dos modelos cascata e espiral, existem outros modelos de desenvolvimento, onde para cada ambiente deve ser utilizado o modelo que melhor se encaixa. Exemplos:

a) desenvolvimento interativo;
b) prototipação;
c) modelo Queda d'Água, entre outros.

Em cada ciclo de vida estão implícitas as validações de funcionalidades implementadas que de maneira obrigatória devem ser executadas, para assim que o sistema desenvolvido obtenha o nível satisfatório de qualidade. A obtenção de tal nível de qualidade torna-se uma tarefa árdua e complexa, onde possui uma forte dependência do conhecimento dos componentes envolvidos no projeto de desenvolvimento.

Hoje a qualidade é fundamental nos softwares desenvolvidos, por tanto as empresas de desenvolvimento estão se dedicando a desenvolver produtos com maior qualidade e menor tempo seguindo padrões e conceitos de qualidade.

2.1 QUALIDADE DE SOFTWARE

A definição de qualidade para Pezzé e Young (2008) é um conceito abrangente que se realiza por intermédio de um conjunto de atividades interdependentes. Existe uma série de atributos associados que refletem na qualidade de software, não sendo somente associados diretamente com o que o software faz, mas sim com o seu comportamento em geral. Tais como: funcionamento, estrutura e organização do programa, tempo de resposta, documentação associada, entre outros (SOMMERVILLE, 2003).

Resumidamente a qualidade vai depender das particularidades do elemento avaliado contra os anseios de quem está avaliando o produto com base em determinadas características. Conforme Pressman (2006) aponta, a qualidade de projeto e a de conformidade, que são os dois tipos de qualidades de software existentes. A primeira remete às propriedades projetadas para determinado item, enquanto que a segunda foca em que nível a equipe de desenvolvedores segue as especificações para o item. Essa distinção também exprime o quanto as equipes de projeto e desenvolvimento podem impactar positivamente ou negativamente no processo de software.

Sommerville (2007) diz que o processo de garantia de qualidade está relacionado à definição e escolha dos padrões aplicados ao desenvolvimento ou ao produto de software, e ainda se a escolha dos métodos a serem utilizados no processo de produção ligados à qualidade é gerencial. Além disso, é importante ressaltar que a produção de software é um processo que envolve, principalmente, seres humanos.

Assim, mesmo com tecnologias de produção de alta qualidade, se a equipe não apresentar qualidade equivalente, de nada adianta.

É comum as empresas de software buscarem a qualidade dos produtos por meio da ampliação de funcionalidades disponíveis a clientes, eliminação de deficiências, cumprimento ou antecipação de prazos, entre outras formas. Porém é necessário ir um pouco além, introduzindo a qualidade no próprio processo produtivo. Para esse anseio algumas etapas precisam ser cumpridas, entre as quais merecem destaque:

a) **garantia e padronização de qualidade:** a qual define padrões para o processo de software direcionais à qualidade desejada (SOMMERVILLE, 2003);
b) **planejamento de qualidade:** em que ocorre a seleção das qualidades a considerar no processo de software e a forma como ocorrerá a avaliação dos quesitos de qualidade escolhidos (SOMMERVILLE, 2003);
c) **controle de qualidade:** que visa garantir a execução dos padrões e procedimentos que regem a qualidade no processo de software (SOMMERVILLE, 2003);
d) **custo de qualidade:** na qual se quantifica o custo para obtenção da qualidade desejada (PRESSMAN, 2006).

Algumas instituições nacionais e internacionais desenvolvem padrões de qualidade cumprindo os requisitos necessários para a qualidade do software desenvolvido, o objetivo é a evolução constante da equipe e o comprometimento das pessoas com o projeto, para assim, garantir a qualidade.

O Software *Engineering Institute* (SEI) do *Carnegie Mellon University* - Estados Unidos, *International Organization for Standardization and the International Electrotechnical Commision* (ISO/EIC) desenvolvem diversas normas e modelos de qualidade de processo de software e dentre elas foram propostas: ISO9001, CMMI, ISO/IEC 15504, entre outros. Dentre os objetivos de tais normas estão apresentar e apontar características que um processo de software deixando a organização livre para sistematizá-la segundo sua própria necessidade.

Segundo o CMMI, existem três tipos de análises sobre os processos organizacionais:

a) **capacidade:** essa análise descreve a gama de resultados que podem

ser atingidos com o uso dos processos;

b) **desempenho:** analisa até o atual estágio dos processos e os resultados obtidos pelo seu uso;

c) **maturidade:** verifica até que ponto um processo especifico está definido, gerenciado, mensurado, controlado e é efetivado. Maturidade implica ter potencial para um crescimento consistente aplicando os processos de software em projetos da organização.

Para se conseguir o que este modelo propõe, a organização deverá evoluir progressivamente, considerando uma sucessão de diferentes níveis. Cada um indica, por sua vez, o grau de maturidade dos processos num determinado instante:

a) **nível 1:** nível onde os processos normalmente estão envoltos num caos decorrentes da não obediência ou ainda, inexistência de padrões;

b) **Nível 2:** nível onde os projetos têm seus requisitos gerenciados neste ponto. Além disso, há um planejamento, a medição e o controle dos diferentes processos;

c) **Nível 3:** nível onde os processos já estão claramente definidos e são compreendidos dentro da organização. Os procedimentos se encontram padronizados, além de ser preciso prever sua aplicação em diferentes projetos;

d) **Nível 4:** nível onde ocorre o aumento da previsibilidade do desempenho de diferentes processos, uma vez que os mesmos já são controlados quantitativamente e estatisticamente;

e) **Nível 5:** nível que tem como objetivo a melhoria continua dos processos, são baseados na compreensão do retorno previsto pela organização e pelo impacto que a mudança ocasionará.

Outra definição importante do CMMI é que áreas de processo também estão subdivididas em categorias, as quais estão distribuídas por vários níveis de maturidade. Assim como outros modelos de processo de software que visam qualidade, o CMMI usa, entre outras rotinas, processos de testes para minimizar ou eliminar possíveis erros do produto. Os testes estão inseridos na área de processo denominada de verificação, que tem como finalidade principal garantir que o produto de trabalho originado em outras práticas especiais como a preparação de testes, verificação do produto final e dos produtos de trabalho intermediários, além de checagens de serviço e sistemas, entre outras práticas (SEI, 2015, tradução nossa).

Considerando que o CMMI comporta também a área de processo de validação, convém destacar que apesar de semelhantes validação e verificação têm objetivos distintos, o primeiro verifica se "você construiu a coisa certa" e o segundo valida se "você construiu direito" (SOMMERVILLE, 2003).

2.2 GARANTIA DA QUALIDADE DE SOFTWARE

Segundo Softex (2009) e Donegan et al (2005), grandes esforços vêm sendo empenhados a fim de melhorar as atividades para garantir a qualidade de um software. De acordo com Sommerville (2007), alguns aspectos devem ser considerados para determinar o nível de qualidade de um produto de software:

a) propósito do software: o grau de confiabilidade oferecido no desenvolvimento deverá seguir de acordo quanto o sistema é critico para organização;
b) expectativas do usuário: é medida por meio dos resultados do uso do software em ambientes e não pelas propriedades do software, em muitas situações, os usuários não ficam surpresos quando ocorre uma falha em um sistema, o que demonstra baixa expectativa em relação a qualidade do produto utilizado (VILLAS BOAS, 2007);
c) ambiente de mercado: para sistemas comercializáveis, deve ser feita uma analise pela organização responsável pelo sistema desenvolvido para analisar estratégias da concorrência, os preços os quais os clientes dispõem a pagar pela solução implementada e o cronograma de entrega deste.

Qualidade de software é considerada um fator estratégico, o conjunto das atividades ligadas à área de garantia da qualidade de software é referenciado como Verificação e Validação e, de acordo com Pressman (2006), contempla: revisões técnicas formais, auditoria de qualidade e configuração, monitoramento de desempenho, simulação, estudo de viabilidade, revisão da documentação, revisão da base de dados, análise de algoritmos e diversos tipos de teste de desenvolvimento, teste de qualificação, teste de instalação entre outros.

O Modelo de Referência MR-MPS do programa de Melhoria do Processo de Software Brasileiro – MPS.BR é dividido em 7 níveis de maturidade apresentados

em uma escala que varia de A (maior maturidade) até G (menor maturidade), e, segundo Softex (2015), cada nível representa um estágio de melhoria dos processos de software de uma organização. O nível D (Largamente Definido) do Modelo de Referência recomenda a definição e implementação dos processos Verificação (VER) e Validação (VAL), considerando a qualidade de software como diferencial de mercado que permitirá às empresas construírem produtos cada vez melhores, a fim de aumentar o nível de satisfação do cliente.

Quando as atividades de garantia de qualidade estão voltadas para manutenção de produtos de software estas costumam ser trabalhosas e difíceis de serem devidamente executadas. Sistemas computacionais evoluem ao longo de seu ciclo de vida. Conforme Pressman (2006), as modificações as quais um sistema é submetido acompanham esse ciclo e ocorrem quando há correção de defeitos, de novas características ou funcionalidades são introduzidas por solicitação do cliente e/ou para atender normas e legislações aplicáveis.

2.3 MODELO DE QUALIDADE MPS.BR

Criado em dezembro de 2003 o MPS.BR é um programa estimulador, coordenado pela Associação para Promoção de Excelência do Software Brasileiro (SOFTEX).

Desenvolvido e mantido pela SOFTEX, o MPS.BR é composto pelo Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e Modelo de Negócio (MN-MPS), recursos com os quais apoia as empresas interessadas em introduzir uma melhoria no processo de software (SOFTEX, 2015).

Destacado como um modelo de qualidade do software o MPS.BR nesse contexto aplicada a validação e verificação que podem ser consideradas um elemento importante para a garantia da qualidade e, portanto, devem ser planejadas e executadas.

A validação de software visa avaliar a qualidade dos componentes ou de um produto completo, sendo a confirmação por meio de evidencia objetiva, onde os requisitos específicos, para um determinado uso pretendido, serão atendidos (ISSO/EIC, 2008).

Fortemente associada à verificação do software a validação pode ser

executado em conjunto com esta, pois muitas vezes torna-se difícil determinar onde começa e onde termina cada uma. De maneira geral pode-se dizer que a validação preocupa-se em assegurar que o produto está sendo desenvolvido de maneira correta como a especificação necessita, ou seja, o produto desejado pelo cliente, já a verificação se preocupa em avaliar se o produto está sendo desenvolvido corretamente conforme os padrões aplicados.

O processo de verificação trata de como avaliar parte ou um produto completo garantindo que atenda a necessidade a seus requisitos, por meio de identificação dos itens a serem verificados, a partir do planejamento da averiguação de cada um desses itens e da execução da validação conforme planejado ao longo do desenvolvimento do software (SOFTEX, 2015).

A partir da validação e verificação de software uma das metas do programa é aprimorar os modelos de melhoria e avaliação do processo de desenvolvimento de software, tendo em vista as micros, pequenas e médias empresas, de forma a atender as necessidades de negócio e ser aprovado nacional e internacionalmente como um modelo de qualidade aplicável a indústria de software.

Figura 2 – Componentes MPS.BR

Fonte: adaptado SOFTEX (2009)

O MR-MPS define níveis de maturidade que são uma combinação entre processos e sua capacidade. A definição dos processos segue os requisitos para uma modelo de referência de processo na ISO/IEC 15504-2, e o alcance de um determinado nível de maturidade se obtêm quando são atendidos os resultados

esperados dos processos e de seus atributos estabelecidos para aquele nível. Os Atributos de Processos (AP) são uma característica mensurável da capacidade do processo. O atendimento aos atributos do processo (AP), pelo atendimento aos resultados esperados Atributos do Processo (RAP), é requerido para todos os processos no nível correspondente ao nível de maturidade. Embora eles não sejam detalhados dentro do processo (SOFTEX, 2015).

O MR-MPS possui sete níveis de maturidade, sendo eles:

a) A - Em Otimização;

b) B - Gerenciado Quantitativamente;

c) C - Definido;

d) D - Largamente Definido;

e) E - Parcialmente Definido;

f) F - Gerenciado;

g) G - Parcialmente Gerenciado.

Cada nível de maturidade possui suas áreas de processos, nas quais os processos e atributos da norma ISO/IEC 12207 são analisados e indicam onde os esforços de melhoria da organização devem ser focados para que atinjam os objetivos do negócio e do modelo de referencia (SOFTEX, 2015).

Os atributos de processos estão divididos em:

a) AP 1.1 - o processo é realizado;

b) AP 2.1 - o processo é gerenciado;

c) AP 2.2 - os produtos de trabalho do processo são gerenciados;

d) AP 3.1 - o processo é definido;

e) AP 3.2 - o processo está implementado;

f) AP 4.1 - o processo é medido;

g) AP 4.2 - o processo é controlado;

h) AP 5.1 - o processo é objeto de melhorias incrementais e inovações;

i) AP 5.2 - o processo é otimizado continuamente.

3 TESTE DE SOFTWARE

O objetivo do processo de teste de software tem como objetivo garantir a qualidade das funcionalidades desenvolvidas, e garantir que cada uma delas corresponda ao que foi solicitado.

A atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificações, projeto e codificação de um software (PRESSMAN, 1995).

A implantação do processo de testes exige investimento em equipes, equipamentos e instalações que geram incertezas de que haverá algum ganho. Contudo o impacto positivo gerado pela adoção do processo de testes é incomensurável, posto que reduz o tempo de produção, diminui custos e amplia a qualidade. As consequências diretas são uma maior produtividade, ampliação na lucratividade, além da melhora na relação com o cliente final pela confiança que o produto e a equipe de produção transmitem (BASTOS et al, 2007).

Myers estabelece uma série de regras que podem servir como objetivo da etapa de teste de um software, tais como:

a) a atividade de teste é um processo de executar um programa como o objetivo de descobrir um erro;
b) um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;
c) um teste bem sucedido é aquele que revela um erro ainda não descoberto.

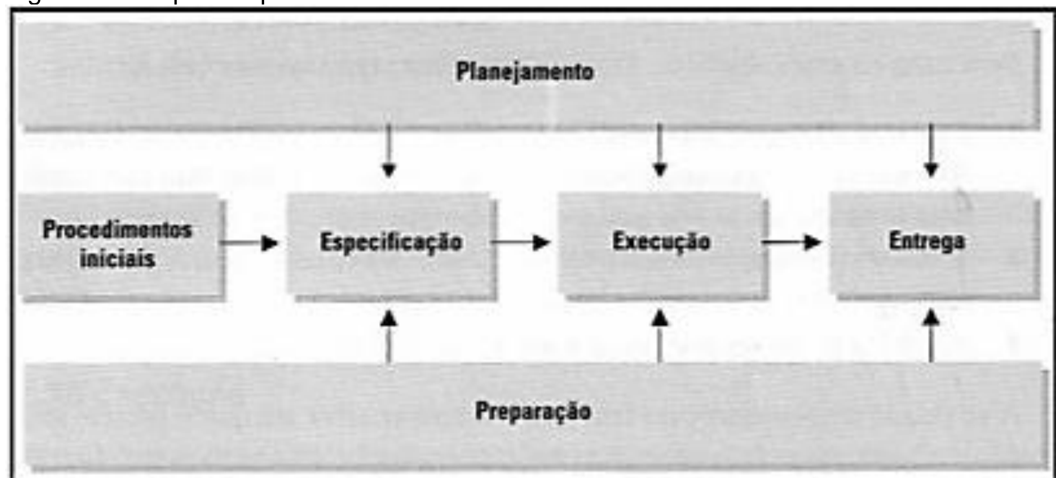
Se uma atividade de teste for conduzida com sucesso, ela tenderá a descobrir erro no software. Como um resultado secundário ao descobrimento de erros, ela também irá demonstrar que as funções para as quais o software foi desenvolvido estão de acordo com as suas especificações funcionais, e estão funcionando corretamente. Porém a atividade de teste não pode mostrar a ausência de erros, ela só pode mostrar que os defeitos estão presentes (PRESSMAN, 1995).

Apesar de eficaz na obtenção da qualidade, as tarefas de teste precisam ser realizadas por equipes especializadas, que conheçam estratégias e técnicas de testes (RIOS; MOREIRA, 2006), além de terem noções sobre regras de negócios que serão convertidas em requisitos funcionais do software. Essas considerações são

importantes porque o nível crítico das rotinas em que o software é empregado determinará a quantidade de testes a realizar (BASTOS et al, 2007).

O teste de software emprega algumas técnicas podendo ser citado entre elas a técnica de caixa-branca, também chamada de estrutural e a técnica de caixa-preta, conhecida como funcional (PRESSMAN, 2006). A realização do processo de teste está dividida em fases ou etapas, que correspondem ao teste de unidade, teste de integração e teste de sistema, Cada uma das etapas possui finalidades especificas, como por exemplo, a etapa dos testes de unidade, que explora as menores particularidades do software ou a de testes de sistemas, na qual o produto é executado na busca de confirmar a exatidão das características do software (DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006; ROCHA; MALDONADO; WEBER, 2001).

Figura 3 – Etapas do processo de teste



Fonte: Rios e Moreira (2006)

A função da fase de teste, segundo Braude (2005), é disponibilizar a entrada ao programa e verificar se a saída é correspondida ao que foi determinado pelos requisitos do software.

Os atributos de qualidade citados por Pressman (2006) são definidos da seguinte maneira:

a) funcionalidade: é avaliada pela observação do conjunto de características e capacidades do programa, generalidade das funções entregues e segurança do sistema global;

b) usabilidade: é avaliada considerando fatores humanos, estética, consistência e documentações globais;
c) confiabilidade: é avaliada medindo-se a frequência e a severidade das falhas, a precisão dos resultados de saída, o tempo médio entre falhas, a capacidade de recuperação de falhas e a previsibilidade do programa;
d) desempenho: é medido pela velocidade de processamento, tempo de resposta, consumo de recursos, vazão e eficiência;
e) suportabilidade: combina a capacidade de estender o programa (extensibilidade), adaptabilidade, reparabilidade - esse três atributos representam em termo mais comum, manutenabilidade – além de testabilidade, compatibilidade, configurabilidade, facilidade com a qual o sistema pode ser instalado e facilidade com a qual os problemas podem ser localizados.

O resultado será a correção de vários defeitos ainda nas documentações ou na modelagem dos sistemas, o que é mais barato, pois evita impactos posteriores no tempo e custo de trabalho, restringindo ocorrências futuras para as seguintes equipes (BASTOS et al, 2007):

a) projetista: evita que novos pedidos de correção tenham que ser estudados e projetados;
b) desenvolvedores: reduz o redesenvolvimento de falhas no projeto em curso ou simplesmente o desenvolvimento do projeto de novos pedidos de correções;
c) equipe de testes: elimina o reteste de falhas do projeto em curso ou simplesmente o teste do projeto de novos pedidos de correções.

Delamaro, Maldonado e Jino (2007) dividem as atividades de teste em fases com objetivos distintos:

a) teste de unidade: aponta erros de programação. Verifica as menores unidades de um programa, podendo ser aplicada a cada nova implementação;
b) teste de integração: verifica a estrutura do sistema, validando as interações efetuadas entre as partes do sistema e se as mesmas estão funcionando corretamente;
c) teste de sistemas: verifica o sistema em um todo após seu termino e se

suas funcionalidades estão de acordo com o que foi documentado.

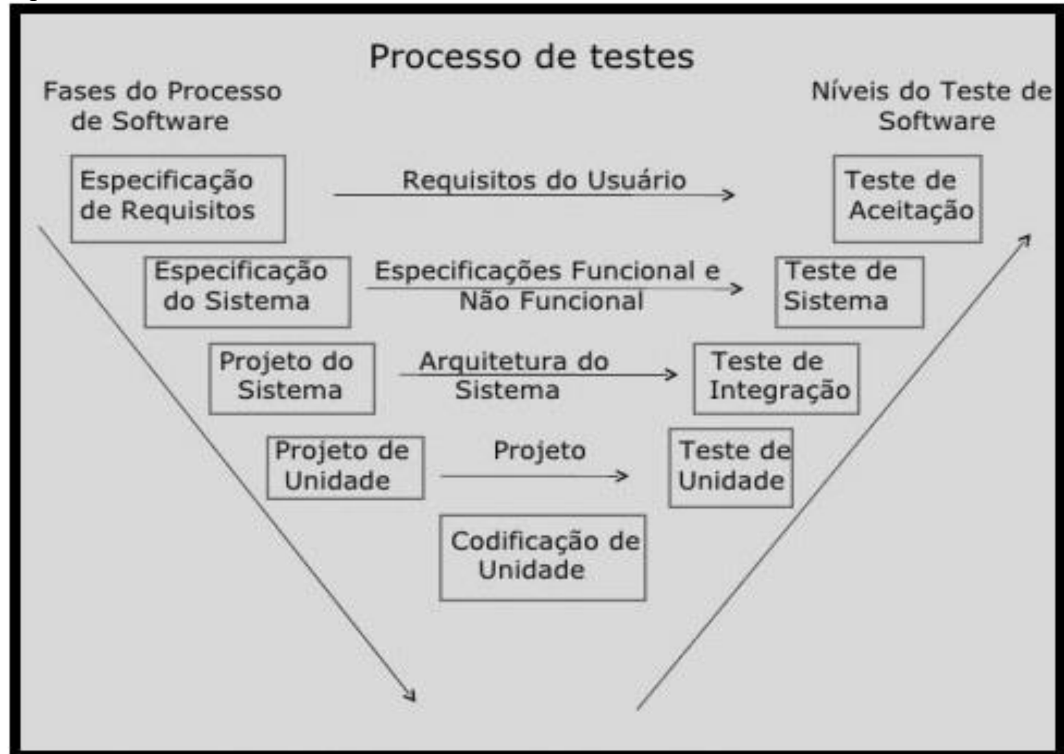
3.1 MODELO “V” DE TESTE DE SOFTWARE

O modelo “V” de testes tem o principal objetivo de prevenir e detectar defeitos e minimizar riscos do projeto enfatizando as atividades de validação e verificação. O ciclo de vida de teste propõe que sejam realizados testes ao longo do desenvolvimento do software, mesmo podendo se dizer que os mesmos são totalmente interdependentes, o clico de testes é dependente da conclusão do software no ciclo de desenvolvimento (BASTOS EI AL, 2007).

Os procedimentos de implementar o sistema e executar os procedimentos de testes devem percorrer juntos até a finalização do desenvolvimento do projeto, com isso podem-se existir dois grupos distintos trabalhando ao mesmo tempo, onde o objetivo de um é implementar o sistema, e o outro tem por objetivo de executar os procedimentos de testes (BASTOS et al, 2007).

A figura 4 mostra os processos de implementação e teste iniciando do mesmo ponto de partida com as mesmas entradas de informações.

Figura 4 – Processo de teste de software



Fonte: Bastos et al (2012)

3.2 TÉCNICAS DE TESTE DE SOFTWARE

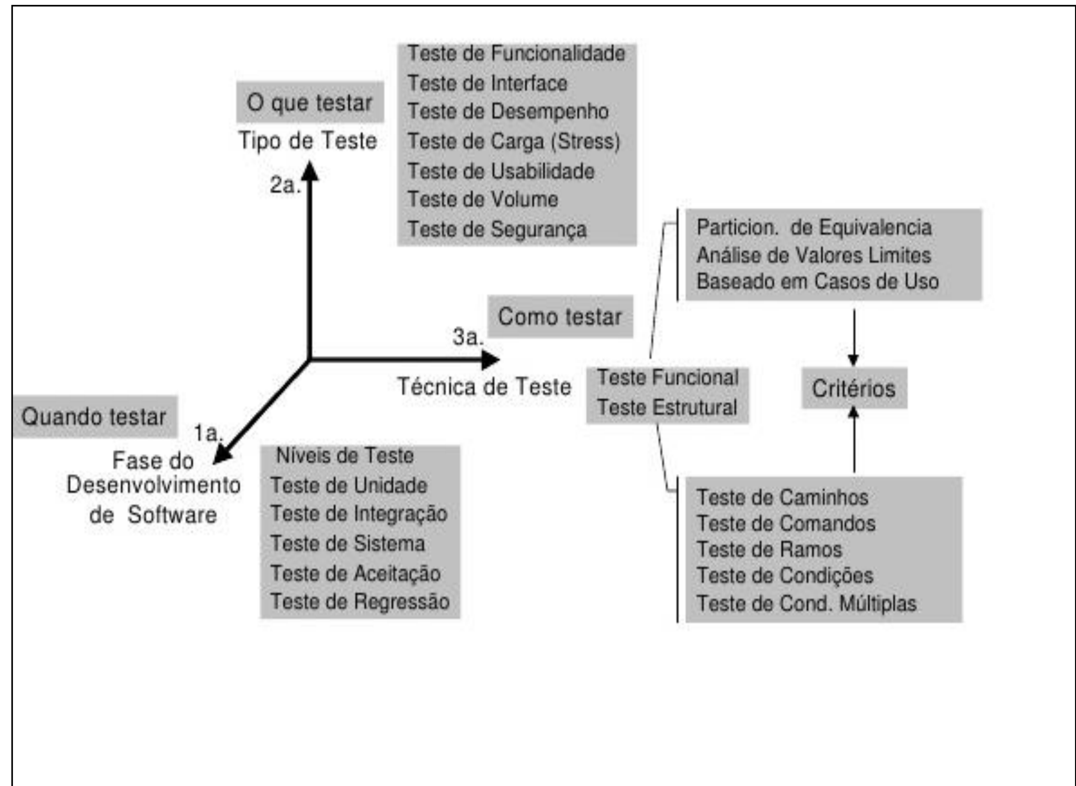
Para garantir a qualidade de um software e para que se possa avaliar os níveis de satisfação perante a análise de requisitos é necessário a utilização de técnicas de teste de software. Conforme Delamaro, Maldonado e Jino (2007) podem ser eficientes na descoberta de defeitos e na garantia de qualidade de software. Várias técnicas podem ser utilizadas para se administrar e avaliar a qualidade.

Levando em questão a complexidade de tamanhos dos softwares desenvolvidos atualmente, fica claro que é muito difícil de atingir 100% de cobertura de todos os requisitos e linhas de códigos existentes. Com isso, as técnicas de testes têm como principal objetivo, auxiliar os analistas a identificar os casos de teste mais importantes, garantindo a maior cobertura possível dentro de um prazo determinado (CAETANO, 2008).

Pezzè e Young (2008) completam dizendo que a escolha das técnicas que serão utilizadas no teste depende dos recursos no desenvolvimento do produto, das restrições de qualidade, custo e prazo de entrega.

Paula Filho (2001) e Pressman (2006) classificam os testes em dois grupos principais: teste de caixa preta e teste de caixa branca.

Figura 5 - Níveis, tipos, técnicas e critérios para definição de estratégias de teste



Fonte: (CRESPO et al, 2004, p.4)

3.2.1 Teste caixa branca

O teste caixa-branca, também chamado de teste estrutural, consiste na técnica em que os casos de testes são criados considerando apenas os recursos lógicos internos do software (MYERS, 2004, tradução nossa).

Bartié (2002) explica que os testes de caixa branca se fundamentam na arquitetura interna do software onde são aplicadas técnicas de teste para localizar defeitos e verificar todas as estruturas utilizadas na codificação.

Através dessa técnica o engenheiro de software consegue derivar casos de teste que garantem os seguintes testes (PRESSMAN, 1995):

a) testar todos os caminhos de códigos, garantindo que cada caminho possível tenha sido exercitado pelo menos uma vez;

b) exercitar todas as decisões lógicas para valores booleanos (falso ou verdadeiro);

c) executar todos os laços em suas fronteiras, dentro de seus limites operacionais;

d) executar estruturas de dados internas garantindo suas validades.

Conforme afirma Pressman (2006) essa técnica possui quatro benefícios:

a) assegurar que dentro de um módulo todos os caminhos tenham sido executados ao menos uma vez;

b) todos os valores falsos ou verdadeiros, decisões lógicas, sejam executados;

c) todos os laços sejam executados;

d) garantem a validade das estruturas de dados internas, as executando.

É importante também destacar que os testes estruturais podem ser afetados por algumas limitações, como por exemplo, os problemas de (DELAMARO; MALDONADO; JINO, 2007):

a) resultados para caminhos distintos: não é possível provar que dois caminhos distintos para fluxo de dados ou de controle podem computar a mesma função;

b) garantia da execução: é indefinível que existam entradas de dados que asseguram a execução de um determinado caminho no fluxo de controle ou de dados do software;

c) caminhos ausentes: é impossível garantir que caminhos ausentes, por falha no projeto ou no desenvolvimento, sejam detectados nos testes;

d) correção coincidente: não há garantias de que uma entrada de dado que gere resultado válido produzirá o mesmo resultado se outro lado for imputado como entrada. Tal problema também pode estar presente em outras técnicas e em diversos tipos de testes

3.2.2 Teste caixa preta

O teste de caixa preta desconsidera a estrutura interna do software, também conhecido como teste funcional, através das entradas fornecidas o teste é

executado assim gerando resultados onde os mesmos são comparados a um resultado esperado definido previamente.

Os métodos de caixa preta têm por principal objetivo ter os requisitos funcionais do software (PRESSMAN, 1995). A estrutura de casos de teste planejada explora, por meio de entradas válidas, as respostas válidas ou inválidas, que demonstram a aderência do software aos requisitos funcionais e ao domínio necessário dos dados. Os objetivos é isolar erros ligados à omissão ou incorreção de funções, mau comportamento de interface, falhas na estrutura ou acesso a dados, problemas de desempenho, bem como erros de iniciação e término (PRESSMAN, 2006).

Bartié (2002) afirma que os grandes benefícios deste método é não necessitar ter conhecimento sobre os códigos fontes utilizados para o desenvolvimento do software ou entender sobre a tecnologia aplicada sobre o mesmo. Para a execução dos testes é necessário apenas conhecer os requisitos básicos e verificar se a especificação está sendo atendida pelo software, o que torna mais fácil a busca por profissionais.

O principal problema para os responsáveis pelos testes é selecionar as principais entradas com maiores probabilidades de encontrar. Muitas vezes essas escolhas são feitas com base na experiência dos engenheiros de software, contudo existem algumas técnicas que podem auxiliar nessas escolhas, tais como (SOMMERVILLE, 2003):

a) particionamento de Equivalência: as entradas válidas ou inválidas que serão submetidas a testes são agrupadas em classes de equivalências.
   Cada classe passa a representar um grupo possuir um determinado comportamento válido ou inválido, os demais são considerados equivalentes a ele. Em suma o número de variações em roteiros ou casos de testes tende a diminuir posto que o teste de um elemento década grupo corresponderá a submeter todo o grupo ao mesmo teste (DELAMARO: MALDONADO; JINO, 2007; PRESSMAN, 2006);
b) análise de Valor Limite: é um critério complementar ao particionamento de equivalência, porém trabalha com áreas limítrofes de domínio para as entradas e saídas. A aplicação do critério de análise do valor limite ocorre sobre as partições de equivalência, produzindo roteiros ou

casos de testes que explorarão os limites de domínio por classes (DELAMARO; MALDONADO; JINO, 2007);

c) grafo causa-efeito: os critérios já apontados apresentam limitações na produção de roteiros ou casos de testes que cubram as reações e comportamentos do software diante de entradas de dados combinadas. Em suma o grafo causa-efeito elimina referências ambíguas e incompletas nas definições dos requisitos funcionais. O uso desse critério requer que o software seja dividido em partes para facilitar a criação do grafo. Posteriormente ocorre a identificação das causas, que seriam entradas válidas ou inválidas, bem como são isolados os efeitos, que correspondem às saídas ou transições de estado, recebendo cada um destes um número correspondente. Após isso ocorre a geração do grafo que define a fluência dos comportamentos do software e nesse momento são documentadas as causas e efeitos com teste impossibilitado. Por último deverá ocorrer a geração de uma tabela de decisão e a partir desta, os casos de testes serão estruturados (MYERS, 2004, tradução nossa);

d) teste de matriz ortogonal: é um critério que preferencialmente deve ser aplicado diante de domínios de entrada que inviabilizam testes exaustivos. A principal vantagem deste critério é permitir a detecção de defeitos oriundos de lógica defeituosa no software (PRESSMAN, 2006);

e) teste de transição de estado: é um critério que testa a lógica do programa e não seus dados. Para modelar esta lógica são usadas máquina de estado, que descrevem o comportamento de um sistema em resposta a estímulos internos ou externos. Para que possam ser usadas nos testes essas máquinas devem modelas a especificação corretamente, ser consistentes e representar de forma precisa os requisitos do sistema que será testado.

Conforme Delamaro, Maldonado e Jino (2007) nos anos 70, métodos para descrever os requisitos e apoiar a especificação de sistemas foram incorporados a técnica de teste funcional. Pressman (2006) e Molinari (2003) dizem que testes funcionais procuram encontrar basicamente os seguintes erros:

a) de interface;

b) no acesso a banco de dados e nas estruturas de dados;

c) de desempenho;

d) de inicialização;

e) funções ausentes ou incorretas.

Completa Pflegger (2004) que o teste funcional deve ter algumas características, tais como:

a) detecção grande de erros;

b) conhecimento das ações e saídas esperadas;

c) uma equipe de teste independente dos projetistas e programadores;

d) testes de entradas validas e invalidas

e) possuir um critério de encerramento.

3.2.3 Teste unitário

Como objetivo principal desta técnica a validação de objetos na fase de desenvolvimento, nas linguagens orientadas a objetos este podem ser as classes desenvolvidas para abranger uma solução (PAULA FILHO, 2001).

Vinculada diretamente a etapa de codificação, o teste unitário é considerado uma associação da mesma, depois que a implementação do código é feita, revisada e verificada, inicia-se o projeto de casos de testes unitários (PRESSMAN, 1995).

3.2.4 Teste funcional

Considerado um teste de caixa preta o teste funcional é uma técnica onde o software é avaliado pelo ponto de vista do usuário, onde são fornecidas as entradas e avaliadas as saídas geradas, verificando assim se estão de acordo com os requisitos funcionais do projeto (DELAMARO; MALDONADO; JINO, 2007).

Os testes funcionais possuem diretrizes que podem ser seguidas para a criação dos casos de testes, tais como (PFLEEGER, 2004):

a) ter grande probabilidade de detectar defeitos;
b) utilizar uma equipe de testes independente dos projetistas ou programadores;

c) ter um critério de encerramento;
d) testar entradas válidas e também as inválidas;
e) não modificar o sistema para facilitar os testes;
f) compreender todas ações e saídas esperadas do sistema;

3.3 GERÊNCIA DE TESTES

Quando um projeto é mal definido ou mal gerenciado torna-se um problema para todos envolvidos com o mesmo, resultando em um desastre financeiro e prejudicando todos envolvidos, mas quando o projeto é bem estruturado seus prazos e desafios podem ser estimulantes e prazerosos (KELLING, 2008 apud SAMPAIO, 2012).

Para que o projeto de testes seja bem sucedido é necessário ter um bom planejamento utilizando de documentação padronizada e que seja guiado por normas internacionais, o principal documento utilizado para guiar os testes é conhecido como plano de testes, e é nele que ficam definidos os níveis de cobertura e abordagens dos testes.

Segundo Bastos et al (2012) a normal IEEE 829:2008 relaciona os seguintes elementos que deverão constar no plano de testes:

Figura 6 – Definição da Norma IEEE 829:2008 para o plano de trabalho.

Introdução:	_Identificador do plano de teste _Escopo _Referências _Nível na sequência de teste _classe de teste e visão das condições de teste.
Detalhes para este nível do Plano de teste:	_Itens de teste e seus identificadores _matriz de rastreabilidade do teste _funcionalidades a serem testadas _abordagem dos testes _critérios de liberação/falha dos itens _requisitos de suspensão e retomada _entregas do teste
Gerencia de teste	_tarefas do teste _necessidades de ambientes _responsabilidades _integração entre as partes envolvidas _recursos e sua alocação _treinamento _cronograma, estimativas e custos. _riscos e contingencias.
Geral	_Procedimentos de garantia de qualidade _Métricas

Fonte: Bastos et al (2013).

As ferramentas de gerenciamento normalmente são utilizadas para fazer a gestão de testes e defeitos. Por exemplo, uma ferramenta que permite que sejam cadastrados os defeitos encontrados no software durante os testes. Essas ferramentas auxiliam a gerenciar quais módulos devem ser testados e a escolher a data de execução, entre outras atividades. Elas são responsáveis por fornecer uma interface entre as ferramentas de execução, por realizar o gerenciamento de defeitos e de requisitos, gerar os resultados e os relatórios de progresso de testes (CUNHA, 2010).

A gestão de defeitos é uma das atividades primordiais de um processo de teste de software. Por meio de uma gestão, é possível acompanhar a qualidade do

software em teste com base nos defeitos cadastrados pelos testadores ao longo de um ciclo de testes. Com base nesses dados é possível identificar áreas problemáticas da aplicação onde os riscos são maiores e planejar atividades preventivas (CAETANO, 2007).

A gestão de defeitos pode ser realizada por meio de ferramentas automatizadas chamadas de *bug tracking system*. Estas ferramentas oferecem um repositório onde os membros da equipe podem cadastrar os defeitos, acompanhar o ciclo de vida destes defeitos e emitir um relatório de gestão.

A gerencia de testes é a parte principal de um processo de testes de software automatizado. Esta atividade é parte importante para o planejamento e controle das atividades de um projeto de testes. A gerencia de testes pode ser feita através de ferramentas de gerenciamento de teste (*test management system*). Estas ferramentas oferecem uma base dados onde os analistas de testes poderão criar casos de testes e atribuir estes a testadores, acompanhar seu status e a execução do caso.

Em decorrência destas necessidades foram desenvolvidas ferramentas de gerência de projetos de software que dão suporte aos usuários, desta forma tentando reduzir o número de falhas que podem ser encontradas durante e após o desenvolvimento de um projeto. Baseado em uma previa pesquisa foram escolhidas duas ferramentas já existentes no mercado, Bugzila*,* Mantins *bugtracker e* TestLink. Estes sistemas podem ser essenciais para o desenvolvimento de um projeto de software.

3.3.1 Bugzilla

Figura 7 – Bugzilla

Fonte: Do autor

Desenvolvido a partir de scripts *Common Gateway Interface* (CGI) Perl o Bugzilla é uma ferramenta construída com base na interface WEB que auxilia no rastreamento de *bugs*. Sendo distribuído sob licença Mozilla Public License e mantido pela Mozila Foundation é uma ferramenta multi-plataforma. O Bugzilla se diferencial das demais ferramentas por seu alto potencial de interagir com sistemas de controle de versão, esta funcionalidade permite que a partir de uma lista de mudanças ocorridas no repositório acessar o *bug*.

Dentre as funcionalidades do Bugzilla podem ser citadas:

a) registro de *bugs*;
b) escalonamento;
c) discussão;
d) relatórios;
e) consultas;

Essas funcionalidades são suportadas por recursos como gerenciador de anexos, integração com e-mail e identificação de conta de usuários. Desenvolvendo um ciclo de vida completo para os *bugs* sendo que este inicia onde o *bug* é reportado e passa por interações de discussão e desenvolvimento até chegar ao ponto final que é “fechado”.

Uma grande desvantagem do Bugzilla é a falta de suporte à documentação colaborativa, onde que para o desenvolvimento de grandes projetos é necessário a interação de uma grande equipe, sendo que este recurso se torna bastante útil.

3.3.2 Mantis *bug tracker*

Figura 8 – Mantis *Bug Tracker*

Fonte: Do autor

Desenvolvido em PHP com suporte a vários bancos de dados como MYSQL e PostgreSQL, e sendo uma ferramenta *Open Source* o Mantis *Bug Tracker* é repleto de recursos completos para suportar o processo de gestão na correção de defeitos encontrados durante o processo de desenvolvimento de software. (ELIAS, 2010).

O usuário pode assumir papeis diferentes dependo do projeto ao qual está vinculado na ferramenta, sendo que em um ele poderá ser o "relator" de defeitos e outro poderá ser o "desenvolvedor" que gerou ou corrigira o defeito encontrado.

Dentre as funcionalidades do Mantis podem ser citadas:

a) Informações da solicitação: Relator (criador da solicitação), categoria (correção, evolução, extração, apoio.), resumo (Informação resumida da solicitação), situação (Status: nova, admitida, atribuída, confirmada, resolvida, fechada), descrição detalhada (Informações da solicitação), prioridades, anexos, etc.;
b) Relacionamentos: Marcação de hierarquia ou duplicidade entre as solicitações, agregação entre solicitações;
c) Anotações: Comentários, ações realizadas, contribuições, dúvidas;
d) Histórico: Registro de movimentação, atualização ou ação executada em

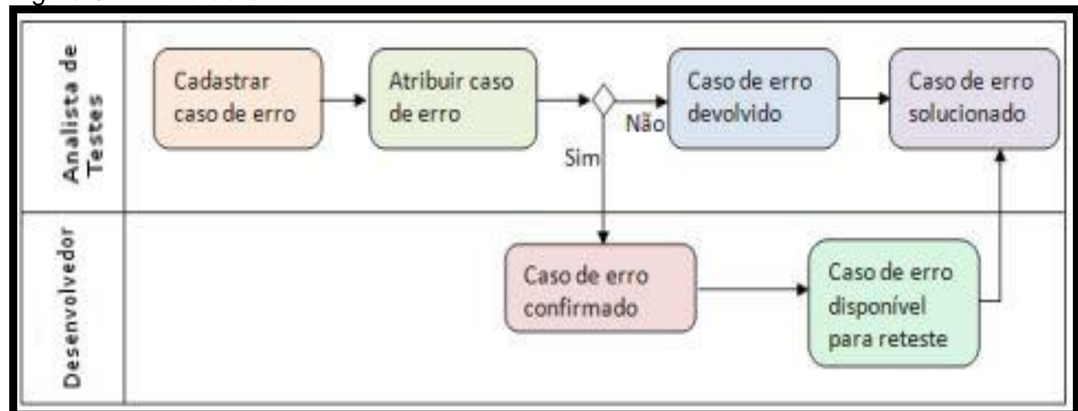
uma solicitação.

Segundo a MantisBT Team com o decorrer do tempo o Mantis *bug tracker* amadureceu e tornou-se popular por ser uma ferramenta simples de gerenciamento de *bugs* e sendo *Opensource* permite aos usuários uma grande variedade de customizações, que inclusive são incentivadas pela empresa do gerenciador no arquivo principal do sistema.

O Mantis possui seis níveis de usuários sendo que cada um deles tem níveis de permissões diferentes que permitem desde apenas a visualizações dos *bugs* cadastrados até o registro dos bugs.

Os *bugs* registrados são assimilados a um usuário do Mantis que automaticamente envia e-mails de notificação informando que esse registro está sobre sua responsabilidade. Para exemplificar o fluxo de erro a figura 9 mostra na visão do analista de testes e do desenvolvedor.

Figura 9 – Fluxo de erro



Fonte: Do autor

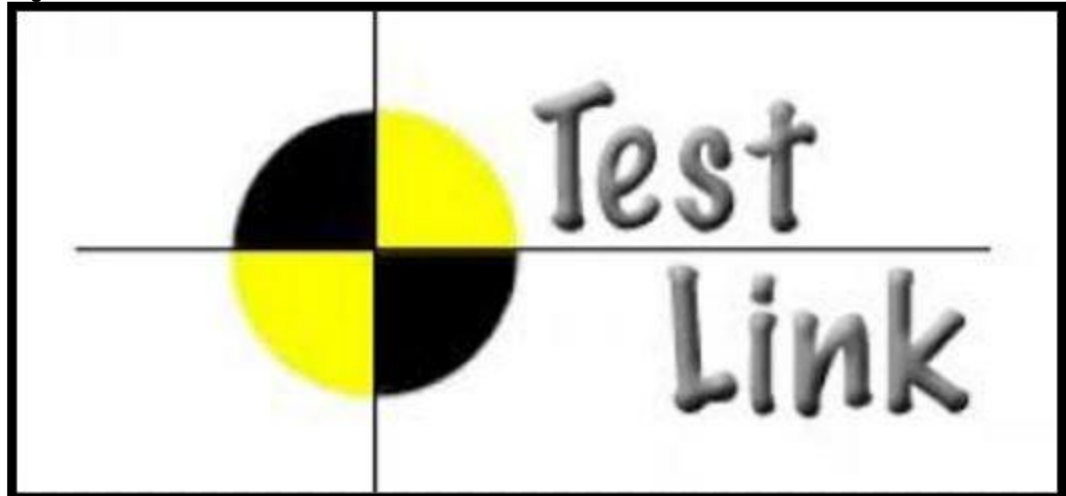
O Mantis possui uma interface gráfica bem amigável, ele disponibiliza visões para acompanhar a evolução e os históricos das mudanças dos *bugs.* Também é disponibilizada uma visão que permite o usuário visualizar as *ISSUES* atribuídas a ele.

A possibilidade de integração do Mantis com outras aplicações é um atrativo da ferramenta, onde ele pode ser integrado com ferramentas de controle de versão, gerenciamento de *bugs* e reportar erros através de outras aplicações através da importação da biblioteca *MantisConnect* no projeto, sendo necessário implementar

os métodos de conexão, sessão, atributos e criação de *ISSUES* para que haja esta integração.

3.3.3 TestLink

Figura 10 – Fluxo de erro



Fonte: Do autor

Mantido pela *Open Comunity Testers* o Testlink e uma ferramenta *Opensource* que tem como objetivo principal auxiliar os processos da fase de produção de um sistema, dependendo da metodologia utilizada. Este é mais utilizado para gerenciamento de casos de uso e organização de planos de testes. A partir desses planos os responsáveis poderão executar casos de teste e consultar resultados de testes dinamicamente, gerar relatórios, rastreamento de requisitos, priorizar e atribuir tarefas (TESTLINK, 2010).

O Testlink pode facilmente integra-se com outras ferramentas que auxiliam no rastreamento de *bugs* e ainda possibilita que os usuários possam adaptá-lo de acordo com suas necessidades podendo ser adicionado, excluído ou alterado as funcionalidades do mesmo.

3.3.4 Análise da aderência das ferramentas no contexto da criação do protótipo

Comparando as ferramentas de gestão de *bugs,* pode-se ser notado que cada uma delas possui particularidades, como a documentação de problemas e não conformidades as ferramentas Bugzilla e Mantis comportam-se de modo equivalente, ambas permitem identificar um defeito e seja feita a documentação do mesmo, já o Testlink não possui módulos que permitam a implementação desse resultado.

Entre o processo de gestão dos *bugs* e acompanhamento as ações até a conclusão das mesmas as ferramentas Bugzilla e Mantis permitem atribuir status para os *bugs* e direciona-los para os responsáveis pela correção do mesmo. Sobressaindo as outras ferramentas o Mantis permite documentar todo o fluxo de status percorrido pelo *bug*.

Em conformidade com o elencado para a criação do protótipo, assegura-se que a ferramenta Mantis é a mais indicada para auxiliar na implementação do mesmo sendo pela facilidade de uso e maior conhecimento do analista de teste sobre esta e considerando que a mesma permite a documentação detalhado do fluxo de testes e contem a API de *webservice SOAP* para cadastramento de erros através de outras aplicações, onde as outras duas ferramentas apenas o Testlink possui a integração para envio dos resultados de execução de scripts de teste.

4 TESTES AUTOMATIZADOS

Antes de decidir automatizar um conjunto de testes, é preciso analisar o software e quais testes serão construídos. O principal motivo para se automatizar um teste é a necessidade de executá-lo diversas vezes. Em um caso típico, existe a necessidade de se executar testes várias vezes. Isso normalmente é suficiente para justificar a automação dos testes (SANTOS, PEDRO, 2009).

De acordo com Donegam et al (2005), testar um produto de software é uma atividade complexa. Fantinato el al (2004) citam fatores que impedem que o teste manual seja executado de forma sistemática, como limitação de tempo e de recursos e o baixo nível de preparo técnico dos profissionais envolvidos, além do nível de complexidade dos sistemas desenvolvidos que vem aumentado ao longo do tempo.

O emprego da automação de teste ganha importância à medida que cresce a busca pela qualidade nos produtos de software ou pela necessidade de se reduzir prazos e custos produtivos. Contudo o uso da automação deve ocorrer somente em empresas de software que possuam uma estrutura madura, contendo uma experiente equipe de testes manuais, conhecedora do produto e de técnicas ou ferramentas adequadas de testes (MOLINARI, 2010; RIOS; MOREIRA, 2006).

Porem a automação de teste de software não pode ser considerado a solução para todos os problemas dos testes de software (MOLINARI, 2010), visto que deverá ocorrer partindo do delicado estudo e análise sobre o que se pretende automatizar, focando em casos de testes reutilizáveis e rotinas e pontos críticos do software que necessitam de validação constante (FEWATER; GRAHAM, 1999, tradução nossa).

Resumidamente pode-se apontar casos de testes mais recomendados para a automação (MOLINARI, 2010):

a) reutilização: compreende todos os casos de testes que podem ser reutilizados, considerando a validade do mesmo em um teste anterior onde o resultado de sua execução tenha resultados válidos;

b) otimização de tempo de teste: abrange todos os casos de testes que exigirão menor tempo para automatizar, considerando estimativas que validem o ganho de tempo que a automação tende a conceder;

c) otimização no uso de recurso: envolve todos os casos de testes que

executados consumirão muitos recursos físicos e pessoais, considerando que este tipo de testes envolve resultados de performance uma atenção especial deverá ser direcionada para o trabalho de automação com o objetivo de viabilizar a ação.

4.1 FERRAMENTAS DE TESTE

Neste contexto é indispensável a utilização de ferramentas de automação de execução de testes. Segundo Molinari (2010) os testes automatizados utilizam uma ferramenta que copia a interação com a aplicação tal qual um humano faria, porém com algumas limitações.

Com diversos focos a automação de testes pode ser aplicada desde controle gerencial a análise de defeitos recorrentes. Cada tipo de automação pode requisitar softwares específicos, entre os quais merecem destaque de ferramentas de (MOLINARI, 2010):

a) plano de testes: compreende softwares com a finalidade de facilitar a projeção e criação de roteiros e casos de testes, alinhados aos requisitos funcionais ou necessidades específicas de teste;
b) automação dos casos de testes: comporta softwares com os quais a equipe de automação grava a execução dos casos de testes. Os softwares com essa finalidade estão divididos em duas categorias, - *Graphical User Interface* (GUI) *Test Drivers* com *Command* Script: onde através de criação de scripts podem ser inseridos comandos de decisão no caso de teste que for gravado,- *Graphical User* Interface (GUI) *Test Drivers* com Visual Script: comportam a criação de scripts a partir da interação gráfica do usuária com sistema sem a possibilidade de criação através de programação;
c) automação de testes de carga e performance: compreende softwares que podem simular a utilização máxima do nível de processamento do software, que exija muitos usuários ou ampla manipulação ou alimentação de dados no sistema;
d) gerência da automação: ferramenta que dá suporte ao gerenciamento dos casos e resultados de testes desenvolvidos;
e) cobertura de código: ferramenta utilizada para avaliar o código

implementado pelo analista de testes, tentando assim verificar toda a cobertura do teste criado e refinar a qualidade do testes desenvolvido;

f) análise de base de conhecimento: a partir das bases de defeitos já existentes proporciona o estudo dos mesmos para criação de novos casos de testes;
g) testes unitários: abrange softwares empregados na automação dos testes unitários, podendo existir um projeto a parte ou a solução estar incorporada no produto.

A aplicação das ferramentas citadas está diretamente relacionada ao foco principal da automação necessitada, como no trabalho atual, que visa demonstrar a união das ferramentas de cobertura de testes funcional e gerência de testes de forma automática.

Dentre as ferramentas dedicadas à gravação e execução de testes automatizados disponíveis no mercado, podem ser destacadas as seguintes:

a) Selenium: ferramenta *Open Source* que é destinada a automação de testes funcionais de softwares *web* (MOLINARI, 2010). Através de gravação e execução de scripts a ferramenta permite comandar um navegador *web* nativo localmente ou remotamente em uma ferramenta que executa os testes em diversos servidores simultaneamente (SELENIUM, 2015);
b) Testcomplete: ferramenta comercial para testes funcionais de software em plataforma desktop e *web*. Através de gravação e execução de scripts cria testes automatizados, possui ferramenta para permitir execução de testes em máquinas que não dispõe do TesComplete instalado e permite a leitura dos objetos de um browser para facilitar a automação. (SMARTBEAR, 2012, tradução nossa);

4.2 MÉTODOS DE AUTOMAÇÃO DE TESTES

De acordo com Molinari (2003) as técnicas de testes funcionais podem estar divididas em dois extremos dentro dos paradigmas do processo de automação:

a) baseados em interface gráfica: Os scripts gerados têm a função de interagir diretamente com a aplicação na interface gráfica, simulando os

passos de um usuário executando as funções de rotina do sistema;

b) baseados na regra de negócio: Neste método os scripts de testes automatizados simulam as funcionalidades da aplicação de um modo onde não haja interação gráfica do teste com a aplicação.

A automação de testes de software como em outros projetos apresenta características comuns como planejamento, análise, implementação e testes (RIOS, MOREIRA, 2006).

Segundo Molinari (2003) existem alguns tipos de automação de testes mais conhecidos:

a) *Capture-Playback*: este método é baseado na interface gráfica da aplicação, sendo considerada mais simples. Através da captura e gravações dos passos executados pelo usuário são gerados scripts contendo os dados de entrada que posteriormente poderão ser reproduzidos quando solicitado;

b) *Data-Driven*: por possuir um alto grau de reutilização este método é considerado mais evoluído. O processo de execução de testes é de forma repetida executando uma mesma ação, porém utilizando dados e entradas diferentes que são retirados e isolados do script e armazenados em uma base de dados externa;

c) *Keyword-Driven*: criada para dar suporte aos testes de aceitação, esta é uma técnica mais avançada com alto investimento de implementação. Os testes são baseados em palavras chaves, a ferramenta de automação gera um conjunto delas sendo que cada uma é considerada um comando de alto nível que representa uma ação a ser executada;

d) *CLI*: baseados em linha de comando essa técnica permite que o usuário possa por meio de um interpretador de comandos do sistema operacional interagir com a aplicação a ser testada.

Para que o processo de automação de testes possa ser iniciado e concluída de maneira eficaz Donegan et al (2005) afirma que documentos básicos como especificação de testes e casos de testes estejam disponíveis, servindo como ponto de partida para que possam ser iniciadas as atividades de testes de software.

5 TRABALHOS CORRELATOS

Alguns trabalhos correlatos foram analisados, sendo que apresentam métodos, técnicas, padrões e assuntos importantes que formam uma base de conhecimento sobre ferramentas de automatização e apoio do processo de testes para o presente trabalho.

5.1 DESENVOLVIMENTO DE UMA MÉTODOLOGIA APLICADA AO GERENCIAMENTO E AOMPANHAMENTO DE TESTES DE SOFTWARE VIA WEB

Trabalho proposto por Fernando Bettiol Lopes para a conclusão de graduação em Ciência da Computação pela Universidade do Extremo Sul Catarinense –UNESC em 2009.

Como principal objetivo o desenvolvimento de uma ferramenta de gerência de testes de software pela técnica funcional, denominado de I&T *Manager*(*Issue* and *Test Manager*). Através do estudo das técnicas de testes funcionais, aplicação dos conceitos de engenharia de teste de software e acompanhamento do ciclo de vida dos testes de software, todo conhecimento absorvido resultou no rastreamento de falhas e aplicação da ferramenta automática para gerenciamento de testes.

Desenvolvido com o uso da linguagem de programação Java, o I&T facilmente poderá ser executado em qualquer sistema operacional ou até diretamente na *web*. Sendo possível mais de um usuário manipular o sistema, cada usuário poderá ter seu perfil armazenado, o I&T comporta as seguintes rotinas:

a) **planejamento de testes:** através do cadastro e ou manutenção de toda rotina de testes como casos, falhas, ciclos e procedimentos;
b) **execução dos testes:** através dos cadastro e ou manutenção dos cronogramas, ou seleção do programa a ser testado, a seleção de casos de testes, ferramentas de testes a serem utilizadas e resultados;
c) **análise dos resultados dos testes:** Gerada através de relatórios dos resultados da execução dos testes, número das métricas de testes, assim como a visualização da fluência nas etapas dos testes;
d) **recursos de depuração:** o I&T apresenta a possibilidade de cadastro de erros conhecidos para a integração com ferramenta de depuração.

Os recursos implementados no I&T por parte de todo o gerenciamento de teste como a contagem de horas de testes, listagem de erros detectados por usuários, aliados a cadastros e parametrizações permitem que este atenda quase que completamente todas as necessidades de um ambiente de testes, trazendo a melhora direta do desempenho do desenvolvimento da equipe ao qual este foi aplicado (LOPES, 2009).

5.2 PROCESSO DE AUTOMAÇÃO DE TESTES DE SOFTWARE COM FERRAMENTA OPEN SOURCE: UM ESTUDO DE CASO COM INTEGRAÇÃO CONTINUA

Trabalho proposto por Cristiana Yukie Masuda para a conclusão de graduação em Sistemas de Informação pela Universidade do Sul de Santa Catarina – UNISUL em 2009.

Com o principal objetivo de garantir a qualidade de software gerando uma padronização nas atividades do processo de testes.

Através da utilização de um ambiente automatizado com apoio de ferramentas open source, houve um maior controle de qualidade sob o software aplicado, onde a definição dos processos de testes, planejamentos dos artefatos e documentos de testes incidiram diretamente neste resultado. Sendo assim os requisitos dos softwares, integração dos testes e identificação das falhas trouxeram conforto a equipe de desenvolvimento perante a qualidade desejada.

O software utilizado para pesquisa um produto de uma empresa de desenvolvimento de softwares de médio porte, juntamente com o desenvolvimento continuo do produto foi criado um projeto de testes ao qual comportaria toda base de conhecimento para criação dos projetos futuros.

Através de indicadores e números a tomada de decisão pode gerar resultados satisfatórios de modo que priorizada a redução de falhas no software e permitindo ser criado um produto da melhor qualidade.

5.3 FRAMEWORK FUNCTEST: APLICANDO PADRÕES DE SOFTWARE NA AUTOMAÇÃO DE TESTES FUNCIONAIS

Essa dissertação foi desenvolvida por Rafael Braga de Oliveira no ano de 2007, como requisito parcial para obtenção do Título de Mestre em Informática Aplicada, na Universidade de Fortaleza, demonstrando um framework para ampliação da reusabilidade e manutenebilidade de suítes de testes automatizados.

O trabalho desenvolvido pelo autor propor um framework para ampliar a reusabilidade e a manutenabilidade de suítes de teste automatizado. A solução foi desenvolvida no Serviço Federal de Processamento de Dados (SERPRO) e utilizada em projetos reais. O framework, denominado FuncTest, utiliza padrões de software e aplica as técnicas *Data-driven* e *Keyword-driven* na estruturação de suítes de teste automatizadas (OLIVEIRA, 2007).

Segundo o autor, um dos principais benefícios observados com o uso de testes automatizados foi a ampliação da cobertura dos testes durante a execução dos testes de regressão, que antes de se desenvolver o framework possuíam escopo bastante restrito e tempo de execução significativamente superior.

5.4 DEF-PRO: APOIO AUTOMATIZADO PARA A DEFINIÇÃO DE PROCESSOS DE SOFTWARE

Este artigo foi desenvolvido por Luis Filipe D. Cavalcanti Machado, Gleison Santos, Káthia Marçal de Oliveira, Ana Regina Cavalcanti da Rocha, sendo apresentado na Universidade Federal do Rio de Janeiro, como objetivo do artigo apoiar a definição do processo de software padrão e uma organização iniciando-se pela definição do processo padrão.

Segundo Machado, Santos, Oliveira e Rocha, a ferramenta Def-Pro tem como objetivo a definição de um processo de software padrão para uma organização com base na Norma ISO/IEC 12207, nos modelos de maturidade, em níveis de capacitação, nas características do desenvolvimento de software da organização e no tipo de ADS para o qual se está definindo o processo. Esta ferramenta de fácil manuseio é de grande utilidade no apoio à definição de processos padrões adequados às organizações e aos padrões internacionais.

5.5 DESENVOLVIMENTO DE UMA METODOLOGIA BASEADA NA AUTOMAÇÃO O PROCESSO DE TESTES DE SOFTWARE COMO ESTRATÉGIA PARA A GARANTIA DA COBERTURA FUNCIONAL

Trabalho proposto por Camilla Damiani para a conclusão de graduação em Ciência da Computação pela Universidade do Extremo Sul Catarinense –UNESC em 2012.

Como principal objetivo o desenvolvimento de uma metodologia baseada na automação do processo de teste de software como estratégia para cobertura funcional, onde a metodologia foi formada principalmente pela criação do ciclo de vida de teste, juntamente com o planejamento da matriz de cobertura e a criação dos testes automatizados. Foram criadas baterias de testes utilizando casos de testes especificados com o conhecimento do especialista. Para a automatização dos testes foi utilizada a técnica de capture-replay na ferramenta TestComplete, que permite a criação e execução de testes de forma rápida e vantajosa.

6 DESENVOLVIMENTO DO PROTÓTIPO PARA AUTOMATIZAR TESTES E REPORTAR DEFEITOS AUTOMATICAMENTE

Com toda pesquisa e levantamento bibliográfico gerado em torno do trabalho desenvolvido o conhecimento adquirido resultou no propósito de desenvolver um protótipo, baseado na união das ferramentas de automação de testes e gerências de testes.

Sendo que por meio de forma automática não haja necessidade de interação humana para reportar as falhas encontradas através dos scripts de testes executados.

O principal objetivo é aumentar a produtividade do analista de testes em relação ao tempo de trabalho, sendo que de forma rápida e eficaz os testes de regressão com scripts já prontos poderão ser executados sem a supervisão humana, e de modo automático onde os erros encontrados serão gravados e poderão ser visualizados pelos responsáveis para correção. Sendo assim realizada a metodologia do presente trabalho.

6.1 METODOLOGIA

Com início no segundo semestre de 2014 onde a proposta do Trabalho de Conclusão de Curso submetido na Universidade do Extremo Sul Catarinense para obtenção do grau de Bacharel em Ciências da Computação foi aprovada pra prosseguir em seu desenvolvimento.

Após aprovação da proposta foi iniciado o projeto de pesquisa que foi constituído do levantamento bibliográfico e elaboração do referencial teórico, utilizando os recursos disponibilizados pela instituição de ensino onde a mesma foi realizada.

Com o levantamento bibliográfico inicial pode-se aprimorar o tema proposto com seus principais aspectos enfatizados, entre estes são possíveis citar as técnicas de testes caixa branca, caixa preta e os meios de gerencia de testes através de ferramentas como Mantis *bug tracker* e Bugzilla.

Para conclusão do trabalho proposto será implementado um protótipo, onde será automatizado o meio de reportar os erros entre os testes automatizados e

as ferramentas de gerência de teste, de forma que não haja necessidade de interação humana.

6.2 SOLUÇÃO A SER TESTADA

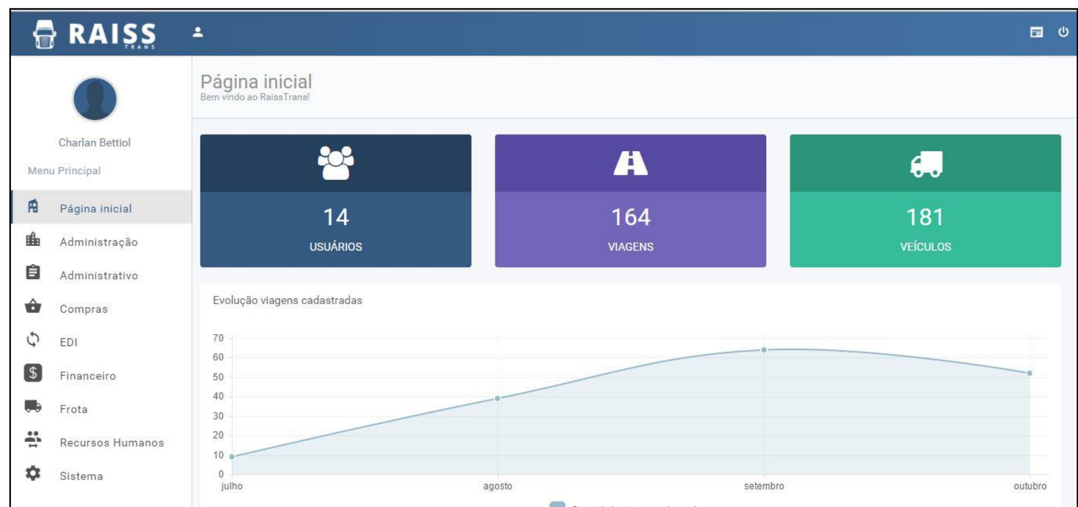
O sistema utilizado para a execução dos testes automatizados desenvolvidos foi o RaissTrans produto desenvolvido pela empresa Raiss Sistemas, o sistema está em fase de desenvolvimento onde os testes automatizados irão auxiliar no ciclo de desenvolvimento da aplicação, onde as rotinas maçantes de alguns testes poderão ser cobertas de forma automática.

O software RaissTrans é uma solução online para gerenciamento de transportadoras, este é aplicado na transportadora automatizando processos, facilitando rotinas e auxiliando os funcionários da transportadora a executar de melhor maneira suas tarefas.

Optou-se pela escolha desse sistema porque o protótipo que será fruto desse trabalho pertence a um integrante da equipe de desenvolvimento da empresa Raiss Sistemas e será continuado ao decorrer do projeto do software.

Abaixo apresenta-se uma imagem da tela inicial do sistema que foi desenvolvido sobre as mais novas tecnologias do mercado tais como AngularJS, Twitter Bootstrap, Java 8, Spring, Rest entre outras.

Figura 11 – Tela inicial do RaissTrans



Fonte: Do autor

6.4 FERRAMENTA DE GERÊNCIA DE TESTES UTILIZADA

A ferramenta de gerência de testes selecionada para o desenvolvimento desse trabalho foi o Mantis *bug tracker,* por ser gratuita e possuir uma API de integração com outros sistemas e plug-ins que serve para customiza-lo e possuir um gerenciamento de bugs, abaixo uma imagem da tela inicial da ferramenta.

Figura 12 – Tela inicial do Mantis bug tracker

Fonte: Do autor

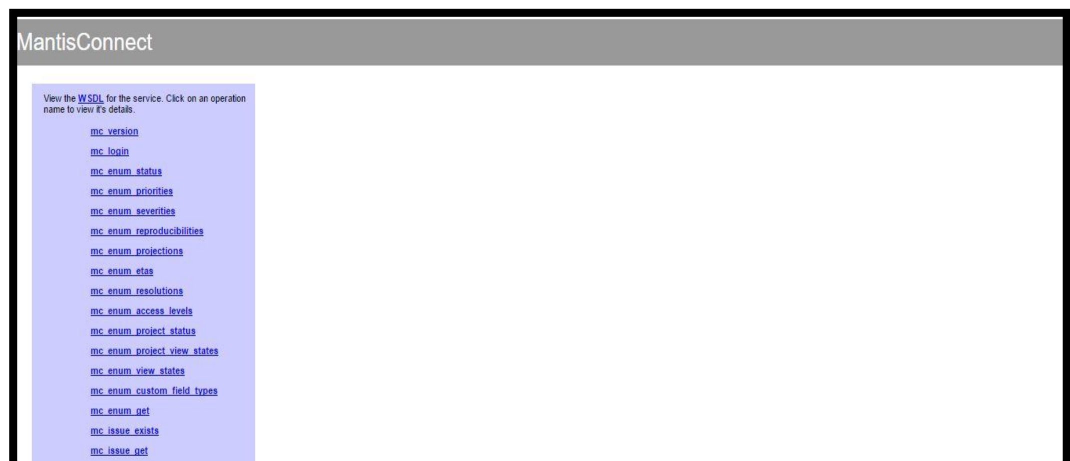
Para o desenvolvimento do protótipo proposto foram utilizados recursos da ferramenta como o recurso de armazenamento de erros que consiste em que possam ser reportados erros na aplicação sendo separados por prioridade, situação atual, a quem está assimilado, como segue abaixo a tela de cadastro de erro

Figura 13 – Tela de cadastro de erros do Mantis bug tracker

Fonte: Do autor

Para a integração dos scripts de testes automatizados com a ferramenta foi utilizado o recurso de integração através de API onde são utilizados serviços SOAP para que possa haver a integração onde o nome do serviço é denominado como *mantisconnect*, através da comunicação por este serviço é possível criar uma sessão entre os scripts e a ferramenta para que dentro desta possam ser reportados os erros, abaixo segue uma imagem do serviço:

Figura 14 – serviço mantisconnect



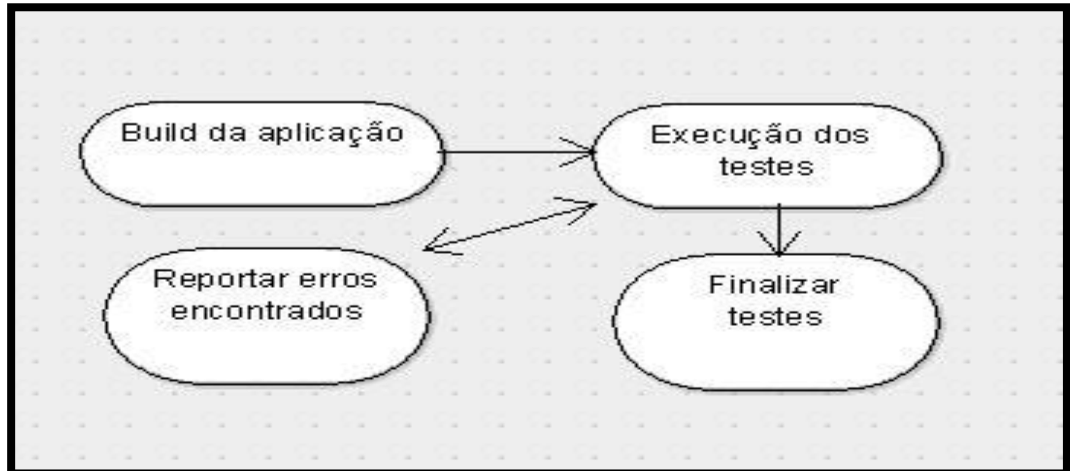
Fonte: Do autor

6.5 DEFINIÇÃO DO CICLO DE VIDA DO PROTÓTIPO

Para organizar as atividades básicas geradas a partir do protótipo, estabelecendo uma precedência e dependência entre as mesmas é necessário a criação de um ciclo de vida dos testes como representa a figura 15 e outro ciclo para

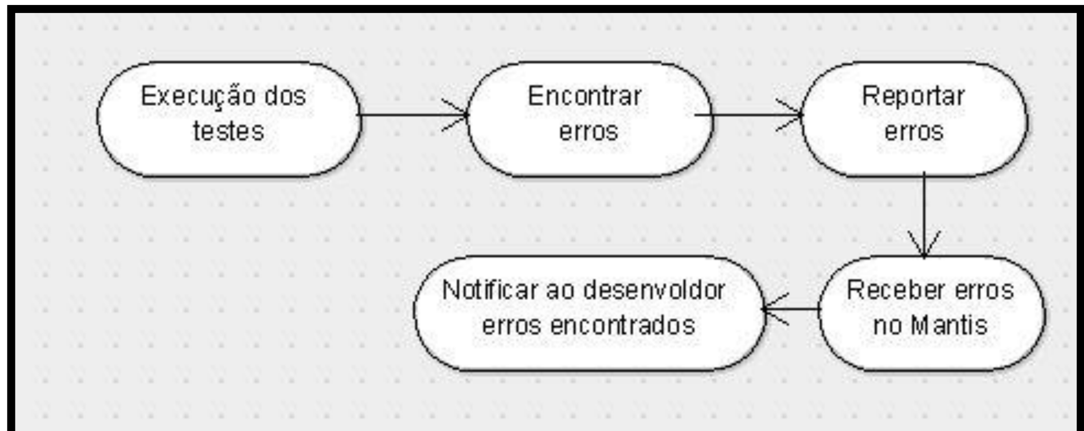
reportar os erros encontrados pelos testes representado pela figura 16.

Figura 15 – Ciclo de vida dos testes



Fonte: Do autor

Figura 16 – Ciclo de vida do protótipo



Fonte: Do autor

A criação do ciclo de vida para as duas rotinas dar-se necessário para estruturar as atividades e suas ordens e dependências.

6.6 BASE DE TESTES

Para que possa ser feita a entrega dos erros ao desenvolvedor é necessário que exista uma base de teste que tenha uma cobertura funcional do requisito levantado na alteração pedida pelo cliente.

Tendo em vista levantar os casos de teste mais adequados para cobrir as funcionalidades que foram afetadas pelo requisito, levando em conta para isso a frequência de manutenção recebidas, bem como o uso constante da rotina.

Para Delamaro, Maldonado e Jino (2007) abranger todos os caminhos de um software ou testar todos os valores de entradas é fundamental, mas como o curto tempo e a necessidade de testes rápido se torna impraticável. Para isto se torna essencial selecionar os requisitos mais importantes e apropriados para criação dos testes.

Com esse intuito foram delimitadas funcionalidades visando a eficiência dos casos de testes e consequentemente o ganho de qualidade. Com a intensão de gerar casos de testes para uma automação funcional regressiva que garanta o funcionamento do software, houve a necessidade de registrar pelo menos um caso de teste para cada funcionalidade conforme figura 17.

Figura 17 – Base de testes

BASE DE TESTES		
Requisito		
Alteração de tipo de dados do cadastro de tanque de combustíveis		
Caso de teste		
ID	Regra de negócio	Sucesso?
CADASTRO DE TANQUES DE COMBUSTÍVEIS		
TC 001	Cadastrar um novo tanque de combústivel	SIM
TC 002	Exibir o tanque de combustível na listagem de tanques	SIM
CADASTRO DE BOMBA DE COMBUSTÍVEL		
TC 003	Vincular um tanque de combústivel a uma bomba de combutível	SIM
VIAGEM ACERTO DE CONTAS		
TC 004	Exibir na lista de Bombas de combustível uma bomba que foi vinculada a um novo tanque	SIM
CADASTRO DE DESPESAS		
TC 005	Vincular um tanque de combústivel a uma despesa	NÃO

Fonte: Do autor

A técnica utilizada para composição da base de teste foi a de gerar conforme conhecimento do especialista. Utilizando a técnica de derivação funcional, os casos de testes foram conduzidos pela ferramenta de automatização Selenium *WebDriver*. A organização da base de teste foi projetada envolvendo as derivações das possibilidades das funcionalidades do software a partir da regra de negócio, onde torna-se necessário elaboras casos de testes para cada componente dependendo do

requisito principal, para que cada um fosse testado individualmente.

Considerando todas características da base de teste observa-se que pode-se ter uma maior cobertura dos testes, aumentando a precisão nos resultados, favorecendo o emprego dos testes funcionais de regressão.

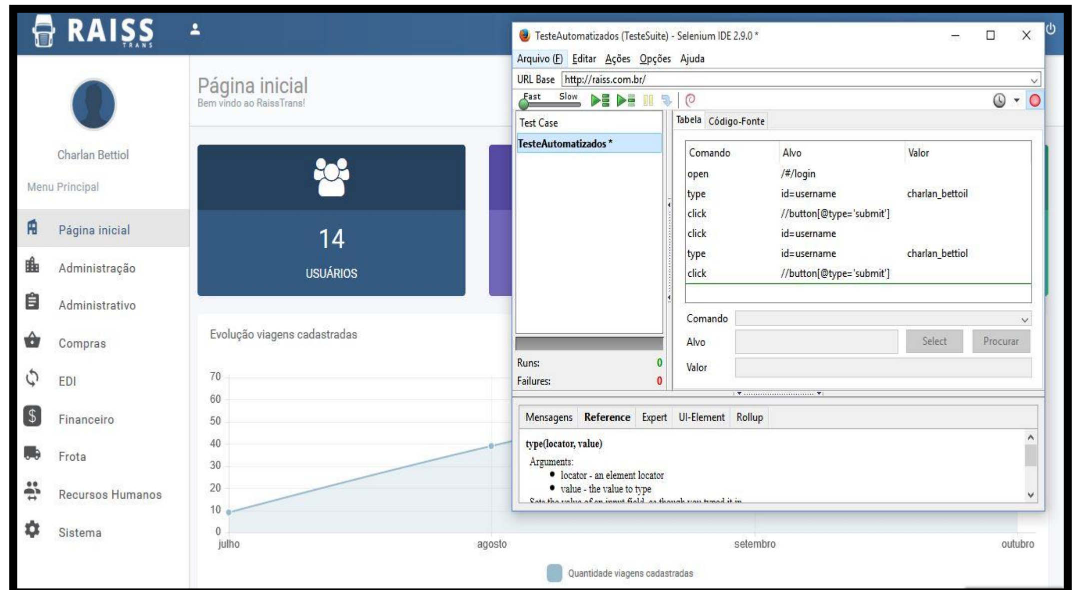
Estruturada a base de testes, o trabalho pode evoluir para a próxima etapa, com a geração dos testes automatizados e reportar os defeitos automaticamente.

6.7 TESTES AUTOMATIZADOS

Para automação dos testes foram empregadas as ferramentas Selenium IDE e Selenium *Webdriver* que são solução para aplicação em software *WEB*, onde para a gravação dos scripts de testes foram realizados testes manuais utilizando manuais utilizando o Selenium IDE em modo de captura, assim fornecendo um script inicial para posterior programação dos scripts de testes desenvolvidos na linguagem Java.

O processo de automação será executado conforme a descrição dos casos de testes gerados a partir do requisito elencado ao qual foi convertido na base de testes. O primeiro passo para a implementação foi a criação de uma suíte de testes na ferramenta Selenium IDE e posteriormente a criação dos casos de testes individuais. Para a gravação dos casos o navegador Mozilla Firefox que é compatível com a extensão da ferramenta é aberto e então é feito o login para que possa ter acesso às funcionalidades elencadas na base de testes conforme figura 18.

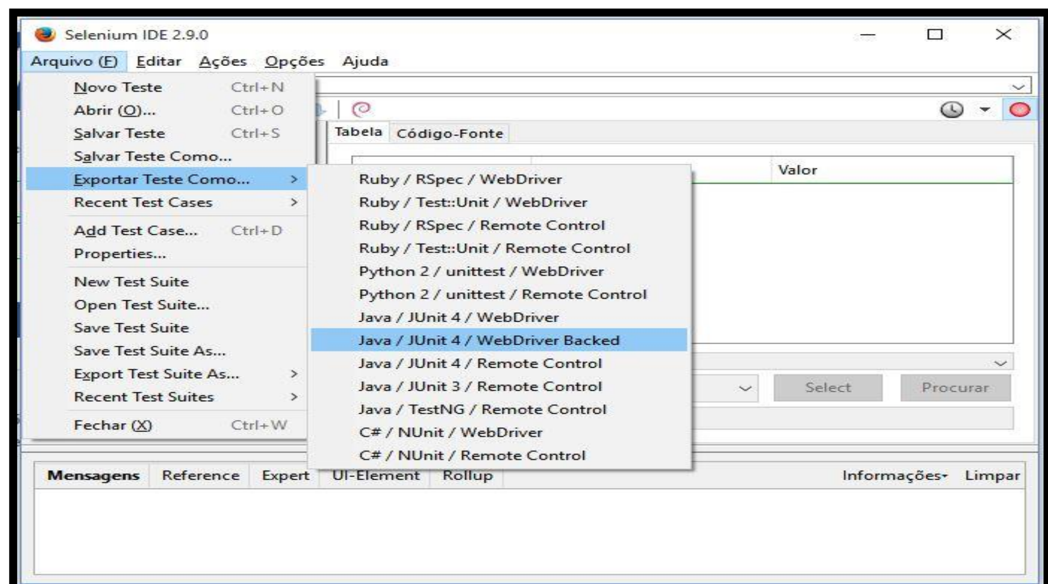
Figura 18 – Script gravados no Selenium IDE



Fonte: Do autor

Após a gravação dos casos de testes, a ferramenta cria um script na linguagem HTML, mas é possível exportar o script criado para a linguagem Java com já com os passos da execução aplicados para a ferramenta Selenium *Webdriver* e *Junit* conforme pode ser visto na figura 19.

Figura 19 – Exportar scripts para Java



Fonte: Do autor

Após exportado o script é gerado um arquivo .java que pode ser aberto com qualquer editor de textos, como para o desenvolvimento do protótipo foi utilizado a linguagem Java o arquivo gerado pelo Selenium IDE foi aberto com o editor de texto e toda parte relacionado a execução dos passos gravados foi copiada para uma IDE desenvolvimento para que o script gerado pudesse ser customizado e utilizado no desenvolvimento dos casos de testes com a utilização dos recursos da linguagem Java, conforme figuras 20 e 21 mostram foi adicionado o recurso *asserTrue* onde nesse simples caso de testes é feita a validação da presença de um texto e comparando esse ao objeto que foi implementado para a verificação da existência do mesmo.

Figura 20 – Script exportado para extensão .java

```
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.regex.Pattern;
import static org.apache.commons.lang3.StringUtils.join;

public class assert tanque cadsatrado com sucesso {
    private Selenium selenium;

    @Before
    public void setUp() throws Exception {
        WebDriver driver = new FirefoxDriver();
        String baseUrl = "http://localhost:8080/";
        selenium = new WebDriverBackedSelenium(driver, baseUrl);
    }

    @Test
    public void testAssert tanque cadsatrado com sucesso() throws Exception {
        selenium.click("//li[9]/ul/li[5]/a/span");
            selenium.type("//input[@type='text']", viagem.getSituacaoViagem().getName());
            selenium.click("xpath=(//button[@type='button'])[2]");
            selenium.click("xpath=(//button[@type='button'])[3]");
            selenium.click("link=Acertar");
            selenium.click("document.acaoAcertarForm.temAbastecimentoInterno[1]");
            selenium.click("//div[@id='acaoAcertarModal']/div/div/form/div[2]/div/div/div/div/div/div/div[2]/label/span");
            selenium.click("//div[@id='acaoAcertarModal']/div/div/form/div[2]/div/div/div[2]/div[2]/div/div/rs-select/div/div/span/i");
            selenium.type("xpath=(//input[@type='text'])[6]", bombaCombustivel.getDescricao());
            selenium.click("css=div.ng-binding.ng-scope");
    }

    @After
    public void tearDown() throws Exception {
        selenium.stop();
    }
}
```

Fonte: Do autor

Figura 21 – Script importado na IDE de desenvolvimento

Fonte: Do autor

Com os scripts importados na IDE de desenvolvimento é possível customizar o código para o melhor aproveitamento da automação dos testes utilizando os recursos da linguagem Java, tendo manutenabilidade do código e sendo fácil a adição de novos scripts de testes para complementar a suíte de testes ou criar novos casos de testes, como a imagem 21 mostra a criação do objeto bombaDeCombustivel que contém todos os atributos necessário de uma bomba de combustível foi feita através da linguagem Java sendo que este objeto já estando preenchido poderá ser utilizado em qualquer parte dos scripts ou suítes de testes ao contrário da ferramenta Selenium IDE que seria necessário a cada script informar o mesmo dado .

Utilizando dos recursos do Java é possível usar todas as validações e recursos que a linguagem oferece e que o Selenium IDE não pode oferecer como *ifs, Asserts* entre outros tipos, além disso seguindo o foco do protótipo esta linguagem oferece a possibilidade de reportar erros de forma automática fazendo a conexão da ferramenta de gerência de testes com os scripts gerados pelo Selenium e já importados e customizados na IDE de desenvolvimento como mostra a abordagem no subcapitulo seguinte.

6.8 REPORTAR ERROS

Com os scripts gravados no Selenium IDE importados na IDE de desenvolvimento e os mesmos já customizados para a automação requisitada pela base de testes tornou-se necessário um meio para reportar os erros encontrados.

Para poder reportar os erros a uma ferramenta de gerência de testes é necessário a criação de um meio de conexão que foi implementado através de uma classe Java onde para cada ferramenta que possua uma API ou meio de conexão e este deve ser desenvolvido de forma diferente e conforme a documentação da própria ferramenta, como neste projeto foi utilizado a ferramenta Mantis *Bug Tracker* foi desenvolvido uma classe Java que se comunica através de uma API de *WEBSERVICE SOAP* conforme exibido na imagem

Figura 22 – Classe de conexão com Mantis

```
public class ConnectMantis implements IConstantes {

    private static ConnectMantis instance = null;
    private static IMCSession sessao = null;

    public ConnectMantis() throws MalformedURLException, MCException {
        URL url = new URL(MANTIS_URL);
        sessao = new MCSession(url, MANTIS_USER, MANTIS_PWD);
    }

    public static ConnectMantis getInstance() {
        if (instance == null) {
            try {
                instance = new ConnectMantis();
            } catch (MalformedURLException ex) {
                Logger.getLogger(ConnectMantis.class.getName()).log(Level.SEVERE, null, ex);
            } catch (MCException ex) {
                Logger.getLogger(ConnectMantis.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        return instance;
    }

    public static IMCSession getSessao() throws MalformedURLException, MCException {
        if (sessao == null) {
            getInstance();
        }
        return sessao;
    }
}
```

Fonte: Do autor

Como a classe de conexão cria uma sessão para reportar os erros encontrados no sistema torna-se necessário ter os dados da conexão como URL, usuário e senha, para isto foi criada uma classe com variáveis constantes que armazenam estes dados para posterior uso na classe de conexão conforme mostra figura 23.

Figura 23 – Classe das constantes de conexão

```
public interface IConstantes {

    /**
     * Constantes dos dados de conexão do mantis
     * URL do servido Mantis, API de conexão
     */
    static final String MANTIS_URL = "http://localhost:8089/mantis/api/soap/mantisconnect.php";

    /**
     * Nome do Usuario para conexão
     */
    static final String MANTIS_USER = "administrator";

    /**
     * Senha do Usuario para conexão
     */
    static final String MANTIS_PWD = "root";

    /**
     * Constantes para o relato do defeito
     */
    static final String MANTIS_PROJETO = "Integracao";
}
```

Fonte: Do autor

Após a conexão estabelecida e os erros encontrados aguardando para serem reportados torna-se necessário a classe para reportar os defeitos, esta utiliza os recursos da linguagem Java e importando a biblioteca de conexão com o Mantis que tem todos os recursos para se reportar uma *ISSUE* e todos os seus atributos necessários, com a biblioteca importada para criar uma nova *ISSUE* é necessário instanciar a classe ISSUE, após isto basta apenas adicionar os atributos da classe compondo uma *ISSUE* como se fosse cadastrada via navegador no Mantis como mostra a figura 24.

Figura 24 – atributos da *ISSUE*

```
sessao = ConnectMantis.getSessao();

// objeto que representa uma issue (bug) no Mantis
Issue issue = new Issue();
issue.setProject(new MCAttribute(1, MANTIS_PROJETO));
issue.setAdditionalInformation(null);
issue.setOs(System.getProperty("os.name"));
issue.setOsBuild(System.getProperty("os.version"));
issue.setPlatform(System.getProperty("os.arch"));
issue.setSeverity(new MCAttribute(70, "crash"));
issue.setReproducibility(new MCAttribute(10, "always"));
LocalDate data = LocalDate.now();
issue.setSummary(sumario + " " + data);
issue.setDescription(descricao);
issue.setCategory(categoria);
issue.setPriority(new MCAttribute(40, "high"));
issue.setAdditionalInformation(informacaoAdicional);
```

Fonte: Do autor

Como a sessão está ativa no momento da criação da *ISSUE* ao ir adicionando os atributos a classe vai enviando esses atributos para o Mantis montando-a, também é possível anexar capturar de tela feitas pelo Selenium como exibi a figura 25.

Figura 25 – Anexar captura de tela

```
long id = sessao.addIssue(issue);
sessao.addIssueAttachment(id, arquivo, "image/png", Base64.decodeBase64(evidencia));
```

Fonte: Do autor

Recebendo essas informações o Mantis monta uma nova *ISSUE,* e exibi ela na sua página inicial as *ISSUES* cadastradas e selecionando uma delas é possível ver os atributos adicionados pelo Selenium através dos testes automatizados.

Figura 26 – *ISSUE* cadastrada no Mantis

View Issue Details [Jump to Notes] [Send a reminder] [Issue History] [Print]

ID	Project	Category	View Status	Date Submitted	Last Update
0000098	Integracao	[All Projects] Automacao	public	2015-10-30 15:26	2015-10-30 15:26
Reporter	administrator				
Assigned To					
Priority	high	**Severity**	crash	**Reproducibility**	always
Status	new	**Resolution**	open		
Platform	amd64	**OS**	Windows 8.1	**OS Version**	6.3
Summary	0000098: testes.TesteBombaCombustivel				
Description	TC005 Vincular tanque a despesa				
Tags	No tags attached.				
Attach Tags	(Separate by ",") Existing tags Attach				
Attached Files	ScreenShot do Erro.jpg [^] (86,713 bytes) *2015-10-30 15:26* [Delete]				

Fonte: Do autor

6.9 NOTIFICANDO O DESENVOLVEDOR

Com os testes automatizados concluídos e toda a base de testes executado e os erros encontrados reportados no Mantis, vem a necessidade de notificar o desenvolvedor dos erros encontrados, a ferramenta de gerência de testes possui uma central de envio de e-mails, mas a mesma deve ser configurada para que o envio seja habilitado, após habilitar o envio deve-se configura-lo internamente no Mantis para que os envolvidos no projeto em que a *ISSUE* foi cadastrada sejam notificados.

Para habilitar o envio de e-mails no *Mantis* deve-se alterar o seu arquivo de configuração *config_defaults_inc* e alterar propriedades como o tipo de envio do e-mail, o servidor SMTP, a porta, o tipo de conexão entre outras configurações, como o desenvolvimento do protótipo deve-se por conta do autor foram usadas as configurações de sua própria conta de e-mail para os envios, abaixo a figura 27 demonstra as configurações.

Figura 27 – Configurações do arquivo no Mantis

```
$g_allow_blank_email     = ON;
$g_limit_email_domain    = OFF;
$g_show_user_email_threshold = NOBODY;
$g_show_user_realname_threshold = NOBODY;
$g_mail_priority         = 3;
$g_phpMailer_method      = PHPMAILER_METHOD_SMTP;
$g_smtp_host             = 'smtp.gmail.com';
$g_smtp_username = 'charlanbettiol@gmail.com';
$g_smtp_password = '************';
$g_smtp_connection_mode = 'tls';
$g_smtp_port = 587;
$g_email_send_using_cronjob = OFF;
$g_email_set_category        = OFF;
```

Fonte: Do autor

Habilitando o envio de e-mails o Mantis apto a enviar notificações para os envolvidos no projeto, mas é necessário por dentro da aplicação configurar o envio, para isto precisa-se acessar o menu de configurações do Mantis que segue o seguinte caminho *MANAGE* > *MANAGE CONFIGURATION* > *E-MAIL NOTIFICATIONS,*

acessado esse menu é possível definir que receberá os e-mail e quais e-mails receberá, para que o sentido do protótipo seja validado é necessário que o desenvolvedor após a execução dos testes automatizados seja notificado caso haja um erro na execução dos mesmo, para que esta notificação ocorra ele precisa ser marcado na coluna *DEVELOPER* com a opção *STATUS CHANGES TO 'NEW',* como mostra a imagem 28.

Figura 28 – Configurações de e-mails no Mantis

In the table below, the following color code applies:
Project setting overrides others.
All Project settings override default configuration.

E-MAIL NOTIFICATION										
Message	User who reported issue	User who is handling the issue	Users monitoring this issue	Users who added Issue Notes	Access Levels					
					viewer	reporter	updater	developer	manager	administrator
E-mail on Change of Handler	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
E-mail on Reopened	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
E-mail on Deleted	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
E-mail on Note Added	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
E-mail on Relationship changed	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Status changes to 'new'	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Status changes to 'feedback'	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Status changes to 'acknowledged'	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Status changes to 'confirmed'	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Status changes to 'assigned'	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Status changes to 'resolved'	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Status changes to 'closed'	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑

Fonte: Do autor

Com as configurações de e-mails feitas no arquivo e habilitadas no Mantis, todas as *ISSUES* cadastradas pelo protótipo notificaram os envolvidos, assim cumprindo o seu papel proposto por esse trabalho, abaixo as imagens 29 e 30 exemplificam as notificações de e-mail.

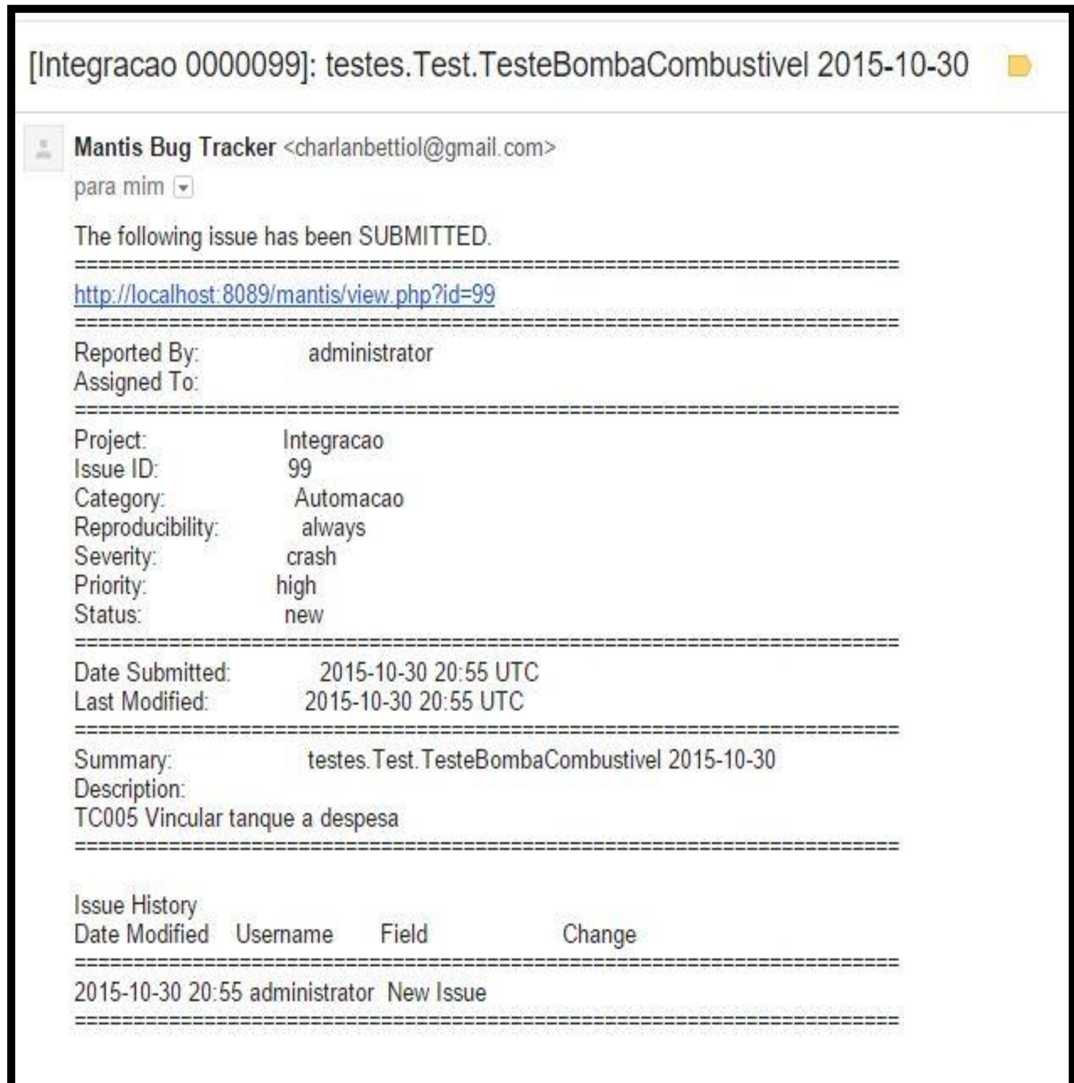
Figura 29 – E-mail recebido na caixa de entrada

Principal | Social | Promoções | +

eu	[Integracao 0000099]: testes.Test.TesteBombaCombustivel 2015-10-30 - The following issue has been SUBMITTED.	30 de out
eu	[Integracao 0000078]: mantis.TesteBombaCombustivel - The following issue has been SUBMITTED.	29 de out
eu	[Integracao 0000077]: mantis.TesteBombaCombustivel - The following issue has been SUBMITTED.	29 de out
eu	[Integracao 0000076]: mantis.TesteBombaCombustivelThu Oct 29 16:45:42 BRST 2015 - The following issue has been SUBMITTE	29 de out
eu	[Integracao 0000075]: mantis.TesteBombaCombustivelThu Oct 29 16:14:23 BRST 2015 - The following issue has been SUBMITTE	29 de out
eu	[Integracao 0000074]: mantis.TesteBombaCombustivelThu Oct 29 14:40:34 BRST 2015 - The following issue has been SUBMITTE	29 de out
eu	[Integracao 0000073]: mantis.TesteBombaCombustivelThu Oct 29 14:32:07 BRST 2015 - The following issue has been SUBMITTE	29 de out
eu	[Integracao 0000072]: mantis.TesteBombaCombustivelThu Oct 29 14:28:01 BRST 2015 - The following issue has been SUBMITTE	29 de out

Fonte: Do autor

Figura 30 – E-mail aberto com todas informações



[Integracao 0000099]: testes.Test.TesteBombaCombustivel 2015-10-30

Mantis Bug Tracker <charlanbettiol@gmail.com>
para mim

The following issue has been SUBMITTED.
==
http://localhost:8089/mantis/view.php?id=99
==
Reported By: administrator
Assigned To:
==
Project: Integracao
Issue ID: 99
Category: Automacao
Reproducibility: always
Severity: crash
Priority: high
Status: new
==
Date Submitted: 2015-10-30 20:55 UTC
Last Modified: 2015-10-30 20:55 UTC
==
Summary: testes.Test.TesteBombaCombustivel 2015-10-30
Description:
TC005 Vincular tanque a despesa
==

Issue History
Date Modified Username Field Change
==
2015-10-30 20:55 administrator New Issue
==

Fonte: Do autor

7 CONCLUSÃO

Dentro da área de qualidade de software, pode-se afirmar que os segmentos de testes sendo bem estruturado e seguindo estratégias e metodologias corretas podem agregar muito valor a um produto.

Embora a automação de testes seja vista como um item de aumento de produtividade no ambiente de testes, as atividades que são aplicadas a esta ainda são aquelas que envolvem a execução de tarefas repetitivas e exaustas facilmente sucessíveis a erros humanos ou difíceis de serem realizadas manualmente. Através de ferramentas especializadas, a automatização surge como um benefício.

Casos de testes de maior complexidade ainda requerem uma execução manual e acompanhamento do profissional de testes, porém existem casos que podem e devem ser automatizados onde apresentam um ganho considerável de produtividades principalmente na relação com o tempo do escopo do projeto, e juntamente com esta automação a integração com a ferramenta de gerência de testes poupando ainda mais o profissional da área de testes, onde o mesmo não precisa ficar assistindo a execução dos testes para reportar os *bugs* encontrados.

Não se tem dúvida que a união da ferramenta de gerencia de testes juntamente com a ferramenta de testes e o meio para reportar defeitos automaticamente é uma solução que alcança agilidade necessária e primordial no ambiente de desenvolvimento. Através do desenvolvimento do protótipo buscou-se diminuir o tempo da fase de testes no escopo de desenvolvimento de um produto e uma maior abrangência nos testes funcionais.

A partir do trabalho apresentado, surgem recomendações para os trabalhos futuros como unir as classes desenvolvidas juntamente com a biblioteca *MantisConnect* para criação de uma única biblioteca, implementar a leitura dinâmica de scripts e configurações do projeto, aplicar o trabalho em uma organização e melhorar a interação dos métodos para reportar erros com os testes e criar uma matriz de rastreabilidade de testes para melhor cobertura funcional.

REFERÊNCIAS

AVIZIENIS, A. et al. **Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing**, v. 1, n.1, p.11–33,2004.Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1335465>. Acesso em: 26/05/2015.

BARTIÉ, Alexandre. **Garantia da Qualidade de Software**. Rio de Janeiro: Elsevier, 2002.

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

BRAUDE, Eric. **Projeto de Software**. Porto Alegre: Bookman, 2005.

BURNSTEIN, Ilene. **Practical Software Testing**. New York: Springer-Verlag New York, Inc., 2010.

CAETANO, Cristiano. **Introdução à Automação de Testes Funcionais**. Disponível em: <http://www.qualister.com.br/blog/introducao-a-automacao-de-testes > . Acesso em: 26/05/2015.

CHRISSIS, Mary Beth, KONRAD, Mike e ASHRUM, Sandy. CMMI – **guidelines for process integration and product improvement**. São Paulo: Pearson Education, 2003.

CUNHA, Simone**. Ambientes de Teste**. Disponível em: <http://testwarequality.blogspot.com.br/p/ambientes-de-testes.html>. Acesso em: 19/05/2015.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario**. Introdução ao teste de software**. Rio de Janeiro: Elsevier, 2007.

DONNEGAN, Paula M.; BANDEIRA, Liane R.P.; MATOS, Ana C.; CUNHA,Paula L.; MAIA, Camila; PIRES, Carlo G.S. **Aplicabilidade da automação de testes funcionais – a experiência no Instituto Atlântico**. Simpósio Brasileiro de Qualidade de Software, 4, 2005, Porto Alegre. Anais. Porto Alegre: Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS, 2005, p. 447-454.

ELIAS, Wagner, **Caracteristicas mantis**. 2010. Disponível em: <http://www.dicas-l.com.br/arquivo/o_uso_de_ferramentas_de_bug_tracker_no_tratamento_de_vulnerabilidadws.php#.UNNpLm_7K1A>. Acesso em: 02/05/ 2015

FANTINATO, Marcelo; CUNHA, Adriano C.R.; DIAS, Sindo V.; MIZUNO, Sueli A.; CUNHA, Cleida A.Q. **AutoTest – um framework reutilizável para a automação de teste funcional de software**. Simpósio Brasileiro de Qualidade de Software, 3, 2004, Brasília. Anais. Brasília: Universidade Católica de Brasília,UCB, 2004, p. 286-299.

FEWSTER, Mark; GRAHAM, Dorothy**. Software Test Automation: effective use of test execution tools**. Inglaterra, Londres: Addison-Wesley, 1999.

MOLINARI, Leonardo. **Inovação e Automação de Testes de Software**. São Paulo: Erica, 2010.

MOLINARI, Leonardo. **Testes de software: produzindo sistemas melhores e mais confiáveis.2 ed**. São Paulo: Érica, 2003.

MYERS, Glenford J**. The Art of Software Testing**. New Jersey: John Wiley & Sons, Inc., 2004.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software: fundamento, métodos e 140 padrões**. 2 ed. Rio de Janeiro: LTC, 2001.

PETERS, James F; PEDRYCZ, Witold, ***Software engineering : an engineering approach.*** USA: John Wiley & Sons, Inc., 2000.

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

PEZZÈ, Mauro; YOUNG, Michal**. Teste e Análise de Software**. Porto Alegre: Bookman, 2008.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Person, 1995.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: McGraw-Hill, 2006.

PRIKLADNICKI, R.; AUDY, J.L.N., **Uma Análise Comparativa de práticas de Desenvolvimento Distribuído de Software no Brasil e no exterior**. XX Simpósio Brasileiro de Engenharia de Software. Porto Alegre, Brasil, 2006.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software: Segunda edição revisada e ampliada**. Brasil: Alta Books, 2006.

SAMPAIO, Marcio Eduardo Correa**. Problemas típicos em gerenciamento de projetos**. Disponível em: <http://www.administradores.com.br/informe-se/artigos/problemas-tipicos-em-gerenciamento-de-projetos-por-marcio-eduardo/20772/>. Acesso em: 18/05/2015

SEI, Software Engineering Institute. **CMMI for Development, Version 1.3. USA: 2010**. Disponível em: <http://www.sei.cmu.edu/reports/10tr033.pdf>. Acesso em: 17/05/2015.

SELENIUM. **Selenium Web Application Testing System**. Disponível em: <http://seleniumhq.org/>. Acesso em: 13/05/2015.

SOFTEX. **2009 ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO** – SOFTEX. MPS.BR – Guia Geral:2009. Disponível em:< https://www.softex.br>. Acesso em: 17/05/2015.

SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Addison-Wesley, 2003.

SOMMERVILLE, Ian. **Engenharia de Software**. 8 ed. São Paulo: Pearson Addison-Wesley, 2007.

VILLAS BOAS, André L. C. **Qualidade e avaliação de produto de software**. Lavras: UFLA/FAEPE, 2007.

WEBAPSEE. **Laboratório de Engenharia de Software da UFPA**. Disponivel em: <http://www3.ufpa.br/webapsee/index.php?option=com_content&view=article&id=47&Itemid=103&lang=br>. Acesso em: 02/05/2015

YOURDON, Edward. **Análise estruturada moderna**. 10. ed. Rio de Janeiro:Campus, 1990. 836p.

TESTE LINK. Disponível em Test Link: Disponivel em: <http://testlink.sourceforge.net/docs/documents/end-users/manual.html>. Acessado em: 02/07/2015

Bugzilla. Disponível em < http://www.bugzilla.org/> Acessado em: 10/07/2015.

ANEXO(S)

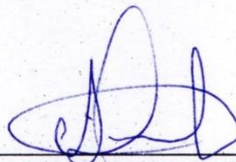
TERMO DE AUTORIZAÇÃO DE USO DE IMAGEM

Por este ato, e na melhor forma de direito, eu, Alan Leite de Rezende , portador(a) de cédula de Identidade R.G. n.º 9073469-5 e inscrito(a) no CPF/MF sob o número 04450587901 , representante legal da empresa Raiss Sistemas , inscrita sob o CNPJ. 23.251.117/0001-43 **AUTORIZO EXPRESSAMENTE** a veiculação, gratuita logomarca/imagem da empresa supramencionada, pelo funcionário Charlan Bettiol, residente na rua Lauro Cirino de Oleira 140, Criciúma, Santa Catarina, inscrita sob o CPF: 083.560.359-88,portador(a) de cédula de identidade 5947478, em quaisquer veículos de comunicação a serem produzidos exclusivamente para a finalidade de conclusão de curso de graduação na Universidade de Extremo Sul Catarinense, em território nacional e internacional.

Para tanto, a imagem objeto da presente autorização poderá ser veiculada por todos os meios de divulgação, inclusive, mas não limitadamente, pela mídia impressa ou por transmissão eletrônica de dados (*online*), em folders de apresentação da entidade, folhetos, malas diretas, bem como no website, através dos quais todo e qualquer terceiro, cliente e/ou visitante, poderá ter acesso às mencionadas informações e imagem, cuja divulgação pública ora se autoriza.

Por ser esta expressão da minha vontade, livre de qualquer constrangimento ou coação, declaro que autorizo o uso acima descrito sem que nada haja a ser reclamado a título de direitos conexos à minha imagem, assinando a presente autorização em duas vias de igual teor e forma.

Criciúma, 09 de Novembro de 2015.



(Alan Leite de Rezende)

APÊNDICE A – ARTIGO CIENTÍFICO

Desenvolvimento de um protótipo para união das ferramentas de automação e gerência de testes de software

Charlan Bettiol[1], Gilberto Vieira da Silva[2]

[1]Acadêmico do Curso Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma - Brasil

[2]Esp. Professor do Curso de Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma - Brasil

charlanbettiol@gmail.com, gilbertovieirasilva@hotmail.com

Abstract. *The software development market is growing rapidly and with this expansion it can be noticed an increasing demand for quality. The companies know that to ensure the software quality is necessary to have a testing team in the development unit to assure the quality regarding the implemented solution. In order to certify the quality the test activities have the crucial role of finding software defects and thereby decreasing the developer's job to correct them later with the solution already running in customer's environment. In this context the test manual execution is fast and effective, but the same implementation in a set of tests and its repetition is an arduous and time consuming task. In this way the testing automation increases the productivity and reaches in a shorter time and with the same effectiveness the cases concerning the repetitious and tiresome tests. To make it easier the communication between these two teams there are specific tools. These are the bug managers, that try to ease the software development process.*

Resumo. *O mercado de desenvolvimento de software cresce rapidamente a cada dia e com esse aumento observa-se uma exigência de qualidade cada vez maior. As empresas sabem que para garantir a qualidade dos softwares é necessário ter presente na equipe de desenvolvimento uma equipe de testes para assegurar a qualidade da solução implementada. Para certificar a qualidade as atividades de teste têm o papel fundamental de encontrar defeitos no software, diminuindo assim o trabalho do desenvolvedor em corrigi-los*

posteriormente com a solução já rodando em clientes. Neste contexto a execução manual de um teste é rápida e efetiva, porém a mesma realização em um conjunto de testes e a repetição deste é uma tarefa árdua e demorada. No sentido em questão a automação de testes aumenta a produtividade e atinge em um tempo menor e com a mesma eficácia os casos de testes mais maçantes e repetitivos. Para facilitar a comunicação entre estas duas equipes, já existem ferramentas especificas, os gerenciadores de bugs que visam facilitar o processo de desenvolvimento de um software.

.

1. Introdução

Atualmente o mercado tecnológico apresenta uma grande demanda de desenvolvimento de softwares, juntamente com esta exigência computacional a busca pela qualidade de software aumenta, em contrapartida os prazos curtos e escopos complexos acabam gerando uma baixa qualidade computacional dos sistemas desenvolvidos. A alta solicitação e necessidade de entregas em curto tempo, gera para as entidades desenvolvedoras a necessidade de buscar um meio para que a entrega seja feita no prazo e com a qualidade desejável pelo cliente final.

As entidades desenvolvedoras junto dos avanços tecnológicos buscam melhores métodos para desenvolver produtos e serviços, diminuindo o tempo, esforço e custos aplicados. Por outro lado, a tecnologia também acrescenta desafios mais complexos ao requerer uma integração do software a sistemas e componentes de terceiros (CHRISSIS; KONRAD; SHRUM, 2007, tradução nossa).

Conforme Pressman (2006), o software possui uma linha evolutiva e apresentará defeitos durante a existência, alguns devido à evolução, pois quanto maior a complexidade, aliada a manutenções constantes, maiores serão as falhas e a queda na qualidade de software.

Visando minimizar os custos e tempo de produção e com a garantia de qualidade final do produto entregue, o uso da automação de testes requer um gerenciamento de testes de software, tendo em vista que é um investimento de longo prazo, que engloba reestruturação e reorganização de equipes e processos, e em alguns casos a aquisição de ferramentas, a produção de alguns impactos também

precisa ser avaliada de forma aproximada, como o aumento de tempo para reportar defeitos encontrados e manutenções de scripts e preparações dos testes (RIOS; MOREIRA, 2006).

Diante dos pontos apresentados o presente trabalho tem por objetivo criar um protótipo para automatizar testes de software e reportar automaticamente as falhas encontradas através da união de ferramentas de gerência de testes e automação de testes, o protótipo deverá garantir que os erros encontrados sejam reportados de forma automática sem a necessidade de interação do analista de testes, garantindo assim a qualidade mínima exigida pelos requisitos e a redução de tempo e custo associados ao desenvolvimento do projeto.

2. Teste de Software

O objetivo do processo de teste de software tem como objetivo garantir a qualidade das funcionalidades desenvolvidas, e garantir que cada uma delas corresponda ao que foi solicitado.

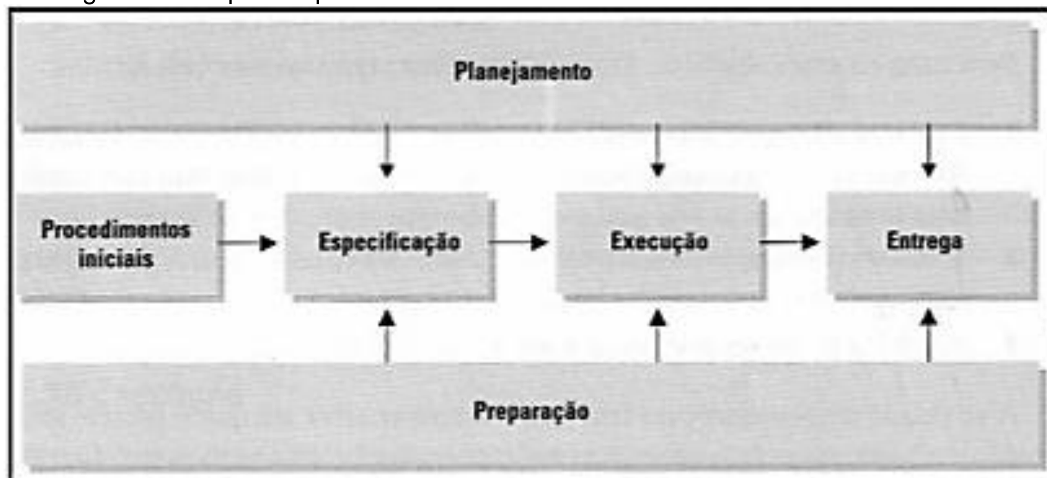
A atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificações, projeto e codificação de um software (PRESSMAN, 1995).

A implantação do processo de testes exige investimento em equipes, equipamentos e instalações que geram incertezas de que haverá algum ganho. Contudo o impacto positivo gerado pela adoção do processo de testes é incomensurável, posto que reduz o tempo de produção, diminui custos e amplia a qualidade. As consequências diretas são uma maior produtividade, ampliação na lucratividade, além da melhora na relação com o cliente final pela confiança que o produto e a equipe de produção transmitem (BASTOS et al, 2007).

O teste de software emprega algumas técnicas podendo ser citado entre elas a técnica de caixa-branca, também chamada de estrutural e a técnica de caixa-preta, conhecida como funcional (PRESSMAN, 2006). A realização do processo de teste está dividida em fases ou etapas, que correspondem ao teste de unidade, teste de integração e teste de sistema, Cada uma das etapas possui finalidades especificas, como por exemplo, a etapa dos testes de unidade, que explora as menores particularidades do software ou a de testes de sistemas, na qual o produto é executado

na busca de confirmar a exatidão das características do software (DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006; ROCHA; MALDONADO; WEBER, 2001).

Figura 1 – Etapas do processo de teste



Fonte: Rios e Moreira (2006)

A função da fase de teste, segundo Braude (2005), é disponibilizar a entrada ao programa e verificar se a saída é correspondida ao que foi determinado pelos requisitos do software.

3. Gerência de Testes

Quando um projeto é mal definido ou mal gerenciado torna-se um problema para todos envolvidos com o mesmo, resultando em um desastre financeiro e prejudicando todos envolvidos, mas quando o projeto é bem estruturado seus prazos e desafios podem ser estimulantes e prazerosos (KELLING, 2008 apud SAMPAIO, 2012).

Para que o projeto de testes seja bem sucedido é necessário ter um bom planejamento utilizando de documentação padronizada e que seja guiado por normas internacionais, o principal documento utilizado para guiar os testes é conhecido como plano de testes, e é nele que ficam definidos os níveis de cobertura e abordagens dos testes.

As ferramentas de gerenciamento normalmente são utilizadas para fazer a gestão de testes e defeitos. Por exemplo, uma ferramenta que permite que sejam cadastrados os defeitos encontrados no software durante os testes. Essas ferramentas auxiliam a gerenciar quais módulos devem ser testados e a escolher a data de execução, entre outras atividades. Elas são responsáveis por fornecer uma interface entre as ferramentas de execução, por realizar o gerenciamento de defeitos e de requisitos, gerar os resultados e os relatórios de progresso de testes (CUNHA, 2010).

A gestão de defeitos pode ser realizada por meio de ferramentas automatizadas chamadas de *bug tracking system*. Estas ferramentas oferecem um repositório onde os membros da equipe podem cadastrar os defeitos, acompanhar o ciclo de vida destes defeitos e emitir um relatório de gestão.

3.1 Bugzilla

Figura 2 – Bugzilla

Fonte: Do autor

Desenvolvido a partir de scripts *Common Gateway Interface* (CGI) Perl o Bugzilla é uma ferramenta construída com base na interface WEB que auxilia no rastreamento de *bugs*. Sendo distribuído sob licença Mozilla Public License e mantido pela Mozila Foundation é uma ferramenta multi-plataforma. O Bugzilla se diferencial das demais ferramentas por seu alto potencial de interagir com sistemas de controle

de versão, esta funcionalidade permite que a partir de uma lista de mudanças ocorridas no repositório acessar o *bug*.

Dentre as funcionalidades do Bugzilla podem ser citadas:

f) registro de *bugs*;
g) escalonamento;
h) discussão;
i) relatórios;
j) consultas;

Essas funcionalidades são suportadas por recursos como gerenciador de anexos, integração com e-mail e identificação de conta de usuários. Desenvolvendo um ciclo de vida completo para os *bugs* sendo que este inicia onde o *bug* é reportado e passa por interações de discussão e desenvolvimento até chegar ao ponto final que é “fechado”.

Uma grande desvantagem do Bugzilla é a falta de suporte à documentação colaborativa, onde que para o desenvolvimento de grandes projetos é necessário a interação de uma grande equipe, sendo que este recurso se torna bastante útil.

3.2 Mantis *bug tracker*

Figura 3 – Mantis *Bug Tracker*

Fonte: Do autor

Desenvolvido em PHP com suporte a vários bancos de dados como MYSQL e PostgreSQL, e sendo uma ferramenta *Open Source* o Mantis *Bug Tracker* é repleto de recursos completos para suportar o processo de gestão na correção de defeitos encontrados durante o processo de desenvolvimento de software. (ELIAS, 2010).

O usuário pode assumir papeis diferentes dependo do projeto ao qual está vinculado na ferramenta, sendo que em um ele poderá ser o "relator" de defeitos e outro poderá ser o "desenvolvedor" que gerou ou corrigira o defeito encontrado.

Dentre as funcionalidades do Mantis podem ser citadas:

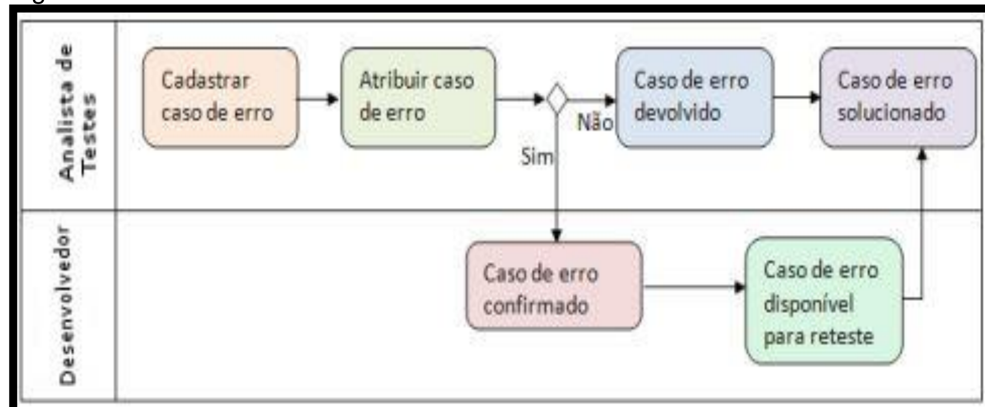
a) Informações da solicitação: Relator (criador da solicitação), categoria (correção, evolução, extração, apoio.), resumo (Informação resumida da solicitação), situação (Status: nova, admitida, atribuída, confirmada, resolvida, fechada), descrição detalhada (Informações da solicitação), prioridades, anexos, etc.;
b) Relacionamentos: Marcação de hierarquia ou duplicidade entre as solicitações, agregação entre solicitações;
c) Anotações: Comentários, ações realizadas, contribuições, dúvidas;
d) Histórico: Registro de movimentação, atualização ou ação executada em uma solicitação.

Segundo a MantisBT Team com o decorrer do tempo o Mantis *bug tracker* amadureceu e tornou-se popular por ser uma ferramenta simples de gerenciamento de *bugs* e sendo *Opensource* permite aos usuários uma grande variedade de customizações, que inclusive são incentivadas pela empresa do gerenciador no arquivo principal do sistema.

O Mantis possui seis níveis de usuários sendo que cada um deles tem níveis de permissões diferentes que permitem desde apenas a visualizações dos *bugs* cadastrados até o registro dos bugs.

Os *bugs* registrados são assimilados a um usuário do Mantis que automaticamente envia e-mails de notificação informando que esse registro está sobre sua responsabilidade. Para exemplificar o fluxo de erro a figura 9 mostra na visão do analista de testes e do desenvolvedor.

Figura 4 – Fluxo de erro



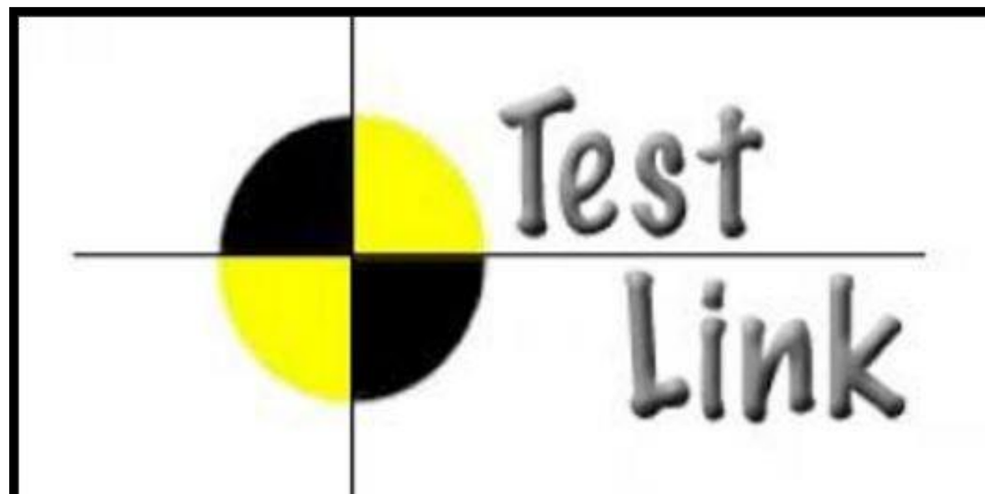
Fonte: Do autor

O Mantis possui uma interface gráfica bem amigável, ele disponibiliza visões para acompanhar a evolução e os históricos das mudanças dos *bugs.* Também é disponibilizada uma visão que permite o usuário visualizar as *ISSUES* atribuídas a ele.

A possibilidade de integração do Mantis com outras aplicações é um atrativo da ferramenta, onde ele pode ser integrado com ferramentas de controle de versão, gerenciamento de *bugs* e reportar erros através de outras aplicações através da importação da biblioteca *MantisConnect* no projeto, sendo necessário implementar os métodos de conexão, sessão, atributos e criação de *ISSUES* para que haja esta integração.

3.3 TestLink

Figura 5 – Fluxo de erro



Fonte: Do autor

Mantido pela *Open Comunity Testers* o Testlink e uma ferramenta *Opensource* que tem como objetivo principal auxiliar os processos da fase de produção de um sistema, dependendo da metodologia utilizada. Este é mais utilizado para gerenciamento de casos de uso e organização de planos de testes. A partir desses planos os responsáveis poderão executar casos de teste e consultar resultados de testes dinamicamente, gerar relatórios, rastreamento de requisitos, priorizar e atribuir tarefas (TESTLINK, 2010).

O Testlink pode facilmente integra-se com outras ferramentas que auxiliam no rastreamento de *bugs* e ainda possibilita que os usuários possam adaptá-lo de acordo com suas necessidades podendo ser adicionado, excluído ou alterado as funcionalidades do mesmo.

4. Testes Automatizados

Antes de decidir automatizar um conjunto de testes, é preciso analisar o software e quais testes serão construídos. O principal motivo para se automatizar um teste é a necessidade de executá-lo diversas vezes. Em um caso típico, existe a necessidade de se executar testes várias vezes. Isso normalmente é suficiente para justificar a automação dos testes (SANTOS, PEDRO, 2009).

De acordo com Donegam et al (2005), testar um produto de software é uma atividade complexa. Fantinato el al (2004) citam fatores que impedem que o teste manual seja executado de forma sistemática, como limitação de tempo e de recursos e o baixo nível de preparo técnico dos profissionais envolvidos, além do nível de complexidade dos sistemas desenvolvidos que vem aumentado ao longo do tempo.

O emprego da automação de teste ganha importância à medida que cresce a busca pela qualidade nos produtos de software ou pela necessidade de se reduzir prazos e custos produtivos. Contudo o uso da automação deve ocorrer somente em empresas de software que possuam uma estrutura madura, contendo uma experiente equipe de testes manuais, conhecedora do produto e de técnicas ou ferramentas adequadas de testes (MOLINARI, 2010; RIOS; MOREIRA, 2006).

4.1. Ferramentas de testes

Neste contexto é indispensável a utilização de ferramentas de automação de execução de testes. Segundo Molinari (2010) os testes automatizados utilizam uma ferramenta que copia a interação com a aplicação tal qual um humano faria, porém com algumas limitações.

Com diversos focos a automação de testes pode ser aplicada desde controle gerencial a análise de defeitos recorrentes. Cada tipo de automação pode requisitar softwares específicos, entre os quais merecem destaque de ferramentas de (MOLINARI, 2010):

h) plano de testes: compreende softwares com a finalidade de facilitar a projeção e criação de roteiros e casos de testes, alinhados aos requisitos funcionais ou necessidades específicas de teste;
i) automação dos casos de testes: comporta softwares com os quais a equipe de automação grava a execução dos casos de testes. Os softwares com essa finalidade estão divididos em duas categorias, - *Graphical User Interface* (GUI) *Test Drivers* com *Command* Script: onde através de criação de scripts podem ser inseridos comandos de decisão no caso de teste que for gravado,- *Graphical User* Interface (GUI) *Test Drivers* com Visual Script: comportam a criação de scripts a partir da interação gráfica do usuária com sistema sem a possibilidade de criação através de programação;
j) automação de testes de carga e performance: compreende softwares que podem simular a utilização máxima do nível de processamento do software, que exija muitos usuários ou ampla manipulação ou alimentação de dados no sistema;
k) gerência da automação: ferramenta que dá suporte ao gerenciamento dos casos e resultados de testes desenvolvidos;
l) cobertura de código: ferramenta utilizada para avaliar o código implementado pelo analista de testes, tentando assim verificar toda a cobertura do teste criado e refinar a qualidade do testes desenvolvido;
m) análise de base de conhecimento: a partir das bases de defeitos já existentes proporciona o estudo dos mesmos para criação de novos casos de testes;
n) testes unitários: abrange softwares empregados na automação dos

testes unitários, podendo existir um projeto a parte ou a solução estar incorporada no produto.

4.2 Métodos de automação de testes

De acordo com Molinari (2003) as técnicas de testes funcionais podem estar divididas em dois extremos dentro dos paradigmas do processo de automação:

a) baseados em interface gráfica: Os scripts gerados têm a função de interagir diretamente com a aplicação na interface gráfica, simulando os passos de um usuário executando as funções de rotina do sistema;
b) baseados na regra de negócio: Neste método os scripts de testes automatizados simulam as funcionalidades da aplicação de um modo onde não haja interação gráfica do teste com a aplicação.

Para que o processo de automação de testes possa ser iniciado e concluída de maneira eficaz Donegan et al (2005) afirma que documentos básicos como especificação de testes e casos de testes estejam disponíveis, servindo como ponto de partida para que possam ser iniciadas as atividades de testes de software.

5. Desenvolvimento da Metodologia

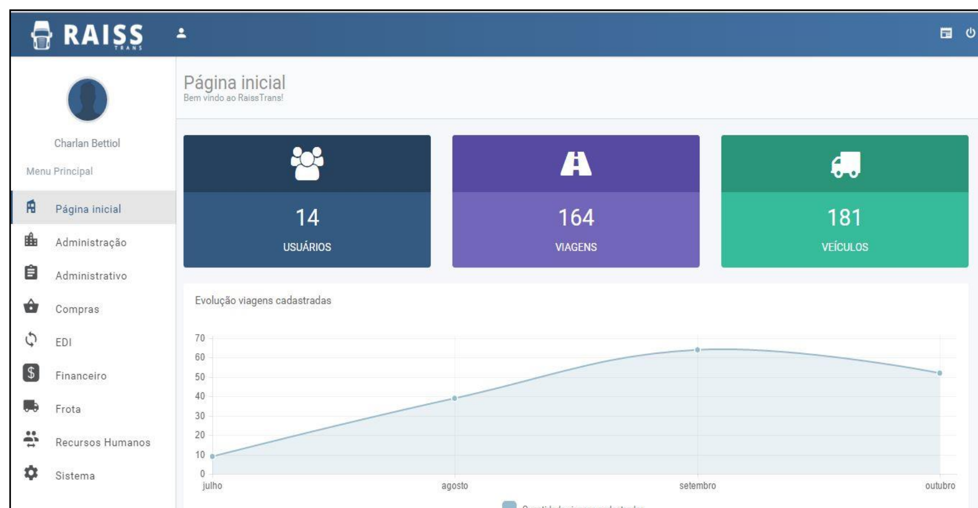
O sistema utilizado para a execução dos testes automatizados desenvolvidos foi o RaissTrans produto desenvolvido pela empresa Raiss Sistemas, o sistema está em fase de desenvolvimento onde os testes automatizados irão auxiliar no ciclo de desenvolvimento da aplicação, onde as rotinas maçantes de alguns testes poderão ser cobertas de forma automática.

O software RaissTrans é uma solução online para gerenciamento de transportadoras, este é aplicado na transportadora automatizando processos, facilitando rotinas e auxiliando os funcionários da transportadora a executar de melhor maneira suas tarefas.

Optou-se pela escolha desse sistema porque o protótipo que será fruto desse trabalho pertence a um integrante da equipe de desenvolvimento da empresa Raiss Sistemas e será continuado ao decorrer do projeto do software.

Abaixo apresenta-se uma imagem da tela inicial do sistema que foi desenvolvido sobre as mais novas tecnologias do mercado tais como AngularJS, Twitter Bootstrap, Java 8, Spring, Rest entre outras.

Figura 6 – Tela inicial do RaissTrans



Fonte: Do autor

5.1 Ferramenta de gerência de testes utilizada

A ferramenta de gerência de testes selecionada para o desenvolvimento desse trabalho foi o Mantis *bug tracker,* por ser gratuita e possuir uma API de integração com outros sistemas e plug-ins que server para customiza-lo e possuir um ótimo gerenciamento de bugs, abaixo uma imagem da tela inicial da ferramenta.

Figura 7 – Tela inicial do Mantis bug tracker

Fonte: Do autor

5.2 Base de testes

Para que possa ser feita a entrega dos erros ao desenvolvedor é necessário que exista uma base de teste que tenha uma cobertura funcional do requisito levantado na alteração pedida pelo cliente.

Tendo em vista levantar os casos de teste mais adequados para cobrir as funcionalidades que foram afetadas pelo requisito, levando em conta para isso a frequência de manutenção recebidas, bem como o uso constante da rotina.

Para Delamaro, Maldonado e Jino (2007) abranger todos os caminhos de um software ou testar todos os valores de entradas é fundamental, mas como o curto tempo e a necessidade de testes rápido se torna impraticável. Para isto se torna essencial selecionar os requisitos mais importantes e apropriados para criação dos testes.

5.3 Testes de automatizados

Para automação dos testes foram empregadas as ferramentas Selenium IDE e Selenium *Webdriver* que são solução para aplicação em software *WEB*, onde para a gravação dos scripts de testes foram realizados testes manuais utilizando manuais utilizando o Selenium IDE em modo de captura, assim fornecendo um script

inicial para posterior programação dos scripts de testes desenvolvidos na linguagem Java.

O processo de automação será executado conforme a descrição dos casos de testes gerados a partir do requisito elencado ao qual foi convertido na base de testes. O primeiro passo para a implementação foi a criação de uma suíte de testes na ferramenta Selenium IDE e posteriormente a criação dos casos de testes individuais.

Após a gravação dos casos de testes, a ferramenta cria um script na linguagem HTML, mas é possível exportar o script criado para a linguagem Java com já com os passos da execução aplicados para a ferramenta Selenium *Webdriver* e *Junit.*

Após exportado o script é gerado um arquivo .java que pode ser aberto com qualquer editor de textos, como para o desenvolvimento do protótipo foi utilizado a linguagem Java o arquivo gerado pelo Selenium IDE foi aberto com o editor de texto e toda parte relacionado a execução dos passos gravados foi copiada para uma IDE desenvolvimento para que o script gerado pudesse ser customizado e utilizado no desenvolvimento dos casos de testes com a utilização dos recursos da linguagem Java.

Com os scripts importados na IDE de desenvolvimento é possível customizar o código para o melhor aproveitamento da automação dos testes utilizando os recursos da linguagem Java, tendo manutenabilidade do código e sendo fácil a adição de novos scripts de testes para complementar a suíte de testes ou criar novos casos de testes.

Utilizando dos recursos do Java é possível usar todas as validações e recursos que a linguagem oferece e que o Selenium IDE não pode oferecer como *ifs, Asserts* entre outros tipos, além disso seguindo o foco do protótipo esta linguagem oferece a possibilidade de reportar erros de forma automática fazendo a conexão da ferramenta de gerência de testes com os scripts gerados pelo Selenium e já importados e customizados na IDE de desenvolvimento.

5.4 Reportar erros

Com os scripts gravados no Selenium IDE importados na IDE de desenvolvimento e os mesmos já customizados para a automação requisitada pela base de testes tornou-se necessário um meio para reportar os erros encontrados.

Para poder reportar os erros a uma ferramenta de gerência de testes é necessário a criação de um meio de conexão que foi implementado através de uma classe Java onde para cada ferramenta que possua uma API ou meio de conexão e este deve ser desenvolvido de forma diferente e conforme a documentação da própria ferramenta, como neste projeto foi utilizado a ferramenta Mantis *Bug Tracker* foi desenvolvido uma classe Java que se comunica através de uma API de *WEBSERVICE SOAP.*

5.5 Notificando o desenvolvedor

Com os testes automatizados concluídos e toda a base de testes executado e os erros encontrados reportados no Mantis*,* vem a necessidade de notificar o desenvolvedor dos erros encontrados, a ferramenta de gerência de testes possui uma central de envio de e-mails, mas a mesma deve ser configurada para que o envio seja habilitado, após habilitar o envio deve-se configura-lo internamente no Mantis para que os envolvidos no projeto em que a *ISSUE* foi cadastrada sejam notificados.

Para habilitar o envio de e-mails no *Mantis* deve-se alterar o seu arquivo de configuração *config_defaults_inc* e alterar propriedades como o tipo de envio do e-mail, o servidor SMTP, a porta, o tipo de conexão entre outras configurações.

6 Conclusão

Dentro da área de qualidade de software, pode-se afirmar que os segmentos de testes sendo bem estruturado e seguindo estratégias e metodologias corretas podem agregar muito valor a um produto.

Embora a automação de testes seja vista como um item de aumento de produtividade no ambiente de testes, as atividades que são aplicadas a esta ainda são aquelas que envolvem a execução de tarefas repetitivas e exaustas facilmente

sucessíveis a erros humanos ou difíceis de serem realizadas manualmente. Através de ferramentas especializadas, a automatização surge como um benefício.

Casos de testes de maior complexidade ainda requerem uma execução manual e acompanhamento do profissional de testes, porém existem casos que podem e devem ser automatizados onde apresentam um ganho considerável de produtividades principalmente na relação com o tempo do escopo do projeto, e juntamente com esta automação a integração com a ferramenta de gerência de testes poupando ainda mais o profissional da área de testes, onde o mesmo não precisa ficar assistindo a execução dos testes para reportar os *bugs* encontrados.

Não se tem dúvida que a união da ferramenta de gerencia de testes juntamente com a ferramenta de testes e o meio para reportar defeitos automaticamente é uma solução que alcança agilidade necessária e primordial no ambiente de desenvolvimento. Através do desenvolvimento do protótipo buscou-se diminuir o tempo da fase de testes no escopo de desenvolvimento de um produto e uma maior abrangência nos testes funcionais.

A partir do trabalho apresentado, surgem recomendações para os trabalhos futuros como unir as classes desenvolvidas juntamente com a biblioteca *MantisConnect* para criação de uma única biblioteca, implementar a leitura dinâmica de scripts e configurações do projeto, aplicar o trabalho em uma organização e melhorar a interação dos métodos para reportar erros com os testes e criar uma matriz de rastreabilidade de testes para melhor cobertura funcional.

REFERÊNCIAS

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

BRAUDE, Eric. **Projeto de Software**. Porto Alegre: Bookman, 2005.

CHRISSIS, Mary Beth, KONRAD, Mike e ASHRUM, Sandy. CMMI – **guidelines for process integration and product improvement**. São Paulo: Pearson Education, 2003.

CUNHA, Simone**. Ambientes de Teste**. Disponível em: <http://testwarequality.blogspot.com.br/p/ambientes-de-testes.html>. Acesso em: 19/05/2015.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario**. Introdução ao teste de software**. Rio de Janeiro: Elsevier, 2007.

MOLINARI, Leonardo. **Inovação e Automação de Testes de Software**. São Paulo: Erica, 2010.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Person, 1995.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: McGraw-Hill, 2006.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software: Segunda edição revisada e ampliada**. Brasil: Alta Books, 2006.

SAMPAIO, Marcio Eduardo Correa**. Problemas típicos em gerenciamento de projetos**. Disponível em: <http://www.administradores.com.br/informe-se/artigos/problemas-tipicos-em-gerenciamento-de-projetos-por-marcio-eduardo/20772/>. Acesso em: 18/05/2015

SELENIUM. **Selenium Web Application Testing System**. Disponível em: <http://seleniumhq.org/>. Acesso em: 13/05/2015.

WEBAPSEE. **Laboratório de Engenharia de Software da UFPA**. Disponivel em: <http://www3.ufpa.br/webapsee/index.php?option=com_content&view=article&id=47&Itemid=103&lang=br>. Acesso em: 02/05/2015

TESTE LINK. Disponível em Test Link: Disponivel em: <http://testlink.sourceforge.net/docs/documents/end-users/manual.html>. Acessado

em: 02/07/2015

Bugzilla. Disponível em < http://www.bugzilla.org/> Acessado em: 10/07/2015.