CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação

BENEFÍCIOS DO TESTE DE SOFTWARE

SILVIA ARANTES PEREIRA

AMERICANA, SP
2013

CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação

BENEFÍCIOS DO TESTE DE SOFTWARE

SILVIA ARANTES PEREIRA

silviapereira.arantes@gmail.com

Trabalho Monográfico, desenvolvido em cumprimento à exigência curricular do Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação da Fatec-Americana, sob orientação do Prof. Me. Anderson Luiz Barbosa.

Área: Engenharia de Software

Americana, SP
2013

(verso da folha de rosto)

(Ficha catalográfica), procurar sempre a Bibliotecária para definir dados específicos da ficha.

Silvia Arantes Pereira

BENEFÍCIOS DO TESTE DE SOFTWARE

Trabalho de conclusão de curso apresentado à Faculdade de Tecnologia de Americana como parte dos requisitos para obtenção do título de Bacharel em Análise de Sistemas e Tecnologia da Informação

Área de concentração: Engenharia de Software.

Americana, dia de mês de defesa da banca de ano.

Banca Examinadora:

_____

Prof. Me. Anderson Luiz Barbosa.

Fatec Americana

_____

Nome completo do membro da banca (Membro)

Maior titulação

Instituição de atuação

_____

Nome completo do membro da banca (Membro)

Maior titulação

Instituição de atuação

Dedico este trabalho a minha família, especialmente aos meus pais Ulisses Lima Pereira e Genessi Arantes dos Santos Lima Pereira, meu amigos, segurança, e chão.

AGRADECIMENTOS

Primeiramente a Deus por toda a força e paciência me concedida em momentos em que o desejo era abandonar o barco. Sua mão sempre se mostrou forte me ajudando a prosseguir. Obrigada, por tirar todas as pedras no caminho, Tú és minha esperança, calma e paz.

Aos meus pais, Ulisses Lima Pereira e Genessi Arantes dos Santos Lima Pereira, que sempre me apoiaram em cada etapa da minha vida, ajudando e incentivando em absolutamente tudo.

Ao meu orientador Professor Anderson Luiz Barbosa pela paciência, dedicação, e prontidão em responder meus e-mails, que muito me ajudaram a prosseguir os estudos nesta área.

A todos os professores da Faculdade de Tecnologia de Americana pela contribuição na minha formação.

Aos amigos e amigas, pelas horas de trabalho em grupo, pelas reuniões semanais, pelos bate-papos e pelos almoços. Especialmente ao Marcelo Santos de Almeida, amigo fiel e companheiro, dedicado aos estudos e esforçado, foi um auxilio importante durante todo o curso, nos intermináveis trabalhos acadêmicos. E aos amigos Isabela Muniz e Samuel Guimarães que fizeram diversos momentos desses quatro anos serem mais leves e suportáveis.

“As coisas podem chegar até aqueles que esperam, mas são somente sobras deixadas por aqueles que lutam.” (Abraham Lincoln).

Resumo

Este trabalho tem como finalidade identificar os benefícios oriundos do teste de software nas organizações que investem recursos nessa área. A importância desse levantamento se dá pelo fato de muitas organizações ainda considerarem desnecessário ter uma equipe de teste de funcionalidade e incluem apenas o teste unitário dentro de seu processo de desenvolvimento, comprometendo a qualidade do que está sendo entregue para o cliente. Para tal, são abordados conceitos de engenharia de software, tipos de testes existentes, automação de testes, ferramentas para automação, uma pesquisa baseada em um levantamento bibliográfico sobre a importância de se testar software, custo de uma equipe de teste e o custo da manutenção do software com defeito, além de uma visão de qualidade de software.

As considerações finais demonstraram que o teste funcional deve ser parte do processo do processo de desenvolvimento de software, uma vez que os gastos para implantar esse teste não são maiores que os gastos equivalentes à manutenção desse software, além dos benefícios ganhos com qualidade, e propõe uma continuação desse trabalho acadêmico visando uma pesquisa de metodologias, que melhore a implantação do teste de software nas companhias de desenvolvimento sem que prejudique a estimativa de novos desenvolvimentos e entrega dentro do prazo prometido para o cliente sem causar gargalo nos testes.

Palavras chaves: Testes; Automação; Qualidade;

Abstract.

This paper aims to raise the benefits from the software testing organizations that invest resources in this area. The importance of this survey is given by the fact that many organizations still consider unnecessary to have a team of functional testing and only include unit testing into their development process, compromising the quality of what is being delivered to the customer . To this end, concepts are addressed some Software engineering , types of existing tests , test automation , automation tools, a search based on literature about the importance of testing software , the cost of a test team and the cost of maintaining the defective software , plus a view of software quality . The final considerations demonstrated that functional testing should be part of the process of software development ,providing the expenses to implement this test , are not larger than equivalent expenses to maintain this software , beyond the benefits gained with quality, and proposes a future academic work towards a research methodology that enhances deployment of software testing in software development without impairing the estimate of new developments and within the promised delivery to the client without bottleneck in the tests.

Key words: Testing, Automation, Quality;

Lista de ilustrações

Sumário

1- INTRODUÇÃO

A construção de produtos com qualidade continua a ser um grande desafio no desenvolvimento de software, é cada vez mais crescente a busca por processos e metodologias que explorem artefatos de engenharia de software a fim de barrar a maior incidência de defeitos possíveis antes que o software chegue ao cliente e, portanto, minimizem o impacto causado por eles.

Segundo uma pesquisa publicada em 2011 pelo instituto *Standish Group* apud *O'keeffe (2012),* feita em empresas dos EUA, foi possível observar que a maioria dos projetos de software iniciados, tem seu tempo estimado esgotado, trazendo prejuízos para ambos os lados da negociação, e o mais preocupante, uma taxa relativamente alta, vinculada ao software, de defeitos, gerando insatisfação dos clientes e um aumento significativo nos gastos envolvendo manutenção do produto.

Neste contexto, a qualidade de produtos de software é um fator crítico de sucesso que deve estar em conformidade com os requisitos, que se tornam base para a construção e padrões de desenvolvimento previamente estabelecidos para o desenvolvimento do produto.

Com base nisso, foram criados diversos processos no modelo de desenvolvimento, como processos de auditorias, chamadas de revisões técnicas formais de código para garantirem que lógicas utilizadas são eficientes e que padrões estabelecidos pela empresa sejam mantidos.

No entanto, mesmo com a criação desses processos, notou-se que muitos softwares com defeitos ainda eram entregues ao cliente e que poderiam causar danos irreparáveis, conforme cita O'Keeffe(2012), em sua dissertação de mestrado, sobre o desfalque sofrido pela empresa *Sainsbury* em 2005 depois do investimento de US$ 526 milhões em um sistema de gerenciamento da cadeia de fornecimento, uma falha não despachava a mercadoria para as lojas em uma rede varejista de comida. Que foi forçada a contratar 3000 funcionários para despachar as mercadorias manualmente *(O'KEEFFE, 2012)*.

Mesmo com os problemas mencionados nesta introdução, muitas empresas ainda consideram desnecessário ter uma equipe de testes de funcionalidade, e incluem apenas o teste unitário dentro do seu processo; Ou ainda quando incluem os testes, sempre ficam para o final do cronograma, sendo a última coisa a ser feita, esmagando os prazos e comprometendo a qualidade.

Diante de tal situação, nasce a necessidade de um levantamento bibliográfico sobre a importância dos testes dentro do processo tradicional de desenvolvimento, de maneira a demonstrar os benefícios que o teste de funcionalidade pode trazer a empresa que o implementa em seu processo.

Portanto, este trabalho acadêmico visa responder a seguinte pergunta: Quais os benefícios que empresas fabricantes de software recebem, ao incluir o teste de software funcional em seu processo de desenvolvimento de softwares comerciais, utilizados por companhias na administração de negócios? Além disso, terá o objetivo central de investigar os benefícios do teste de software para a qualidade do produto entregue ao cliente, dentro do processo de desenvolvimento de software. E os objetivos específicos de explicar o conceito de teste de software e os tipos de testes existentes, descrever os mecanismos de testes de software, usados para automatizar testes e explorar as ferramentas que são utilizadas por testadores.

Inicialmente este estudo foi baseado em pesquisa bibliográfica, seleção de artigos científicos em revistas e Internet, leitura de pelo menos uma monografia pertinente ao assunto escolhido. E em complemento a esta pesquisa será realizado um levantamento comparativo de custos para manter uma equipe de teste e corrigir um defeito.

O trabalho está organizado da seguinte forma: no capítulo 2 tem se uma revisão de literatura, onde será apresentado um pouco sobre engenharia de software, especificamente os modelos de desenvolvimento de software e serão abordados conceitos como erro, defeito e falha. Será realizada também uma introdução ao teste de software, e quais os tipos de teste existentes.

No Capítulo 3 serão abordados os temas automação de testes e ferramentas utilizadas durante o processo de teste. No capítulo 4 serão apresentadas os benefícios do uso de teste como parte do processo de desenvolvimento de software, e no capítulo 5 serão apresentados as considerações finais.

2. INTRODUÇÃO AO TESTE DE SOFTWARE

Ao contrário do que se imagina a preocupação em testar software não é nova, embora a maioria das metodologias de desenvolvimento de software seja recente. Em 1957, Charles L. Baker fazendo um estudo sobre o livro Digital *Computer Programming*, estabelece uma distinção entre eliminar defeitos e testar software. Em 1961 o livro *Computer Programming Fundamentals* apresenta um capítulo sobre teste de software. Em 1969, Dijkstra usa essa a afirmação "Teste mostra a presença e não a ausência de defeitos" em uma conferência para o comitê OTAN na Itália (outubro de 1969). Em 2009, a Microsoft tinha nos seus quadros mais de 10 mil testadores (Apud Emerson Rios 2013) [1].

Já os modelos de desenvolvimento de software apareceram na década de 80. Tais modelos, cascata, espiral, entre outros surgiram com a ideia de organizar e inserir padrões no processo de desenvolvimento de software.

Dentro do processo de construção, estão presentes as atividades de desenvolvimento e testes. Foi a partir desta organização que o conceito de testes de softwares tornou-se mais conhecido.

Em 1990 os fornecedores de software passaram a integrar em uma suite única, ferramentas para testes funcionais e de gerenciamento. E a partir do ano de 1995 surgiram as primeiras ferramentas para automação de testes e de *performance* para algumas plataformas.

No ano de 2002, surgiu no Brasil a Associação Latino Americana de Teste de Software -ALATS[2]

> A Associação Latino Americana de Testes de Software (ALATS 2013), é uma associação sem fins lucrativos, que busca divulgar as boas práticas em Teste de Software. Temos como propósito suportar a comunidade de testes, valorizando o desenvolvimento técnico e científico de novos métodos e processos, visando aumento na produtividade e a melhoria da qualidade

[1] Disponível em< http://www.emersonrios.eti.br/Artigos/Historia%20resumida%20em%20fatos%20do%20Teste%20de%20Software%20resumida.pdf > acessado em 20 ago. 2013

[2] Disponível em< http://www.alats.org.br/portal/missao-proposito.html> acessado em 21 ago. 2013

> dos produtos desenvolvidos. Buscamos uma maior integração entre as universidades, empresas e profissionais de TI, facilitando o desenvolvimento e aplicação de técnicas para atender a atual demanda e necessidades do mercado de Testes de Software.
>
> (ALATS, 2013)

Atualmente a ALATS promove palestras, encontros mensais e seminários. Dentre os eventos, o mais conhecido é o Seminário Brasileiro de Teste de Software (BRATESTE). Este seminário abrange assuntos como processos de testes, técnicas, ferramentas, entre outros.

2.1. Qualidade de software

A qualidade de um software é uma área que objetiva garantir a qualidade do software através da definição e normatização de processos de desenvolvimento. Apesar dos modelos aplicados na garantia da qualidade de software atuar principalmente no processo, o principal objetivo é o de garantir um produto final que satisfaça às expectativas do cliente, dentro daquilo que foi acordado inicialmente.

De acordo com Koscianski (2007), no desenvolvimento de software, a qualidade do produto está diretamente relacionada à qualidade do processo de desenvolvimento, dessa forma, é comum que a busca por um software de maior qualidade passe necessariamente por uma melhoria no processo de desenvolvimento.

Dentre os processos criados para garantirem a qualidade do produto entregue, destacam-se os processos independentes de testes para a validação dos módulos desenvolvidos. Estes testes adiantam o cenário dos problemas no software, logo, os gastos envolvidos para resolver esse tipo de problema será menor. Esse processo como qualquer outro depende de pessoas especializadas, chamadas de testadores e é feito manualmente por cada um deles. O modelo de teste foi introduzido de uma forma muito positiva, trazendo benefícios para as empresas desenvolvedoras do software, confiança para a empresa contratante e abrindo novas fronteiras para novo modelo de desenvolvimento, como por exemplo, os testes automáticos. Segundo a ISO 9000, 2000[3], “qualidade é um grau em que um

[3] Disponível em <http://www.qualidade.eng.br/artigos_iso9000.htm >. Acessado em 30. Ago. 2013

conjunto de características inerentes a um produto, processo ou sistema cumpre os requisitos inicialmente estipulados para estes".

2.1.1. Modelos de qualidade

Em 1977, McCall *apud* Martins (2012) propôs um modelo para a avaliação da qualidade de software. Esse modelo envolve um conjunto de três fatores que avaliam o software com relação a três pontos de vista distintos.

> I-Com relação ao uso do produto (Características Operacionais).
>
> Correção: o quanto um programa satisfaz a sua especificação e cumpre os objetivos visados pelo cliente.
>
> Confiabilidade - À medida que se pode esperar que um programa execute sua função pretendida com a precisão exigida.
>
> Eficiência - É a quantidade de recursos computacionais e de código exigida para que um programa execute sua função, com total precisão, visando realizar a operação de forma 100% segura.
>
> Integridade - Medida na qual, controla-se o acesso ao software e aos dados, bloqueando assim o acesso de pessoas não autorizadas, para que não ocorra perda de dados ou de código.
>
> Usabilidade - Mede a facilidade para a utilização do software.·.
>
> II - Com relação à alteração do produto (Habilidade para ser alterado).
>
> Manutenção - O esforço exigido para localizar e reparar erros em um programa.
>
> Flexibilidade - O esforço utilizado para realizar uma alteração no software, isto é, qual o grau de facilidade que o software oferece para a sua alteração, de forma rápida e eficaz.
>
> Testabilidade - São todos os recursos utilizados, no teste do software, isto é, o esforço exigido para testar um programa a fim de garantir que ele execute a função pretendida.
>
> III - Transição do produto (Adaptabilidade a novos ambientes).
>
> Portabilidade - Mede a facilidade com que um produto pode ser movido para outra plataforma, ou software.
>
> Reusabilidade - Medida na qual o software, ou parte dele, pode ser reusado em outros softwares, em outras palavras, o código do software deve ser reaproveitável.
>
> Interoperabilidade - O software é capaz de ser acoplado ao outro.
>
> (MARTINS, 2012, pág. 20).

Conforme mencionado na citação anterior, dizer que um software é excelente em qualidade significa dizer que todos os caminhos, tanto em relação ao uso do produto, como em relação à alteração do produto e transição do mesmo, foram percorridos e executados corretamente. O teste de software consegue validar muitos dos pontos de qualidade, como correção, testabilidade, entre outros. Porém a qualidade de um software depende de um trabalho em equipe bem documentado, de uma verificação completa dos recursos que serão consumidos, e das demandas que este software atenderá. Além de normativas e padrões de qualidade que cada segmento em questão deve seguir.

2.2. Conceitos básicos associados à engenharia de software

O teste ou verificação e validação, é parte integrante da engenharia de software, presente em diversos modelos de desenvolvimento. Portanto se torna importante o conhecimento de alguns conceitos dessa ciência.

2.2.1. **Defeito, erro e falha**

Para compreender a finalidade do teste de software, é preciso conhecer alguns conceitos importantes ligados a essa atividade. Primeiramente será importante entender a diferença entre Defeitos, Erros e Falhas. A Figura 1, ilustra essa diferença.

> **Defeito** é uma imperfeição de um produto. O defeito faz parte do produto e, em geral, refere-se a algo que está implementado no código de maneira incorreta. (**KOSCIANSKI**, 2007, p. 31)
>
> **Erro** é uma manifestação concreta de um defeito num artefato de software. Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução de um programa constitui um erro. (**ARILO NETO,** 2007 p. 55).
>
> **Falha** é o comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ter sido causada por diversos erros e alguns erros podem nunca causar uma falha. (**Arilo Neto,** 2007 p. 55)

Figura 1 - Erro, defeito e falha.
Fonte: **Arilo Neto,** 2007 p. 55

2.2.2. Ciclo de vida do software

Segundo Pressman (1995), a engenharia de software compreende um conjunto de etapas que envolvem métodos, ferramentas e procedimentos. Essas etapas são demonstradas através dos ciclos de vida de um software. Alguns deles serão apresentados nesse trabalho, com o objetivo de demonstrar a presença do teste de software dentro desses ciclos e, portanto a sua importância dentro do processo de desenvolvimento.

Neste trabalho também serão apresentados alguns dos modelos de processos de software existentes.

2.2.2.1. Modelo em cascata

O modelo em cascata é o primeiro modelo de ciclo de vida de um software, segundo Sommerville (2007), ele foi criado dos processos mais gerais da engenharia de sistema. Por se tratar de um processo custoso para as empresas, devido aos custos de produção, uma vez que neste modelo uma fase não pode ser iniciada antes que a outra termine, ou seja, quando existe um problema na fase de requisitos, por exemplo, ou teste, as primeiras fases precisam ser refeitas gerando retrabalho e custo elevado, conforme a Figura 2 - Modelo em Cascata. Esse modelo, quase não é mais usado pelas empresas de desenvolvimento de software, porém seu beneficio é a geração completa de artefatos de software, o que ajuda no teste, e na manutenção do software. É um Modelo que:

> “Considera as atividades fundamentais do processo, compreendendo especificação, desenvolvimento, validação e evolução, a as representa como fases de processos separadas, tais como especificação de requisitos, projeto de software implementação e teste”. (SOMMERVILLE, 2007, p. 43)

Note que nesse processo, os testes iniciam-se na fase de implementação, com os testes unitários e se estendem até a fase de integração com o teste unitário.

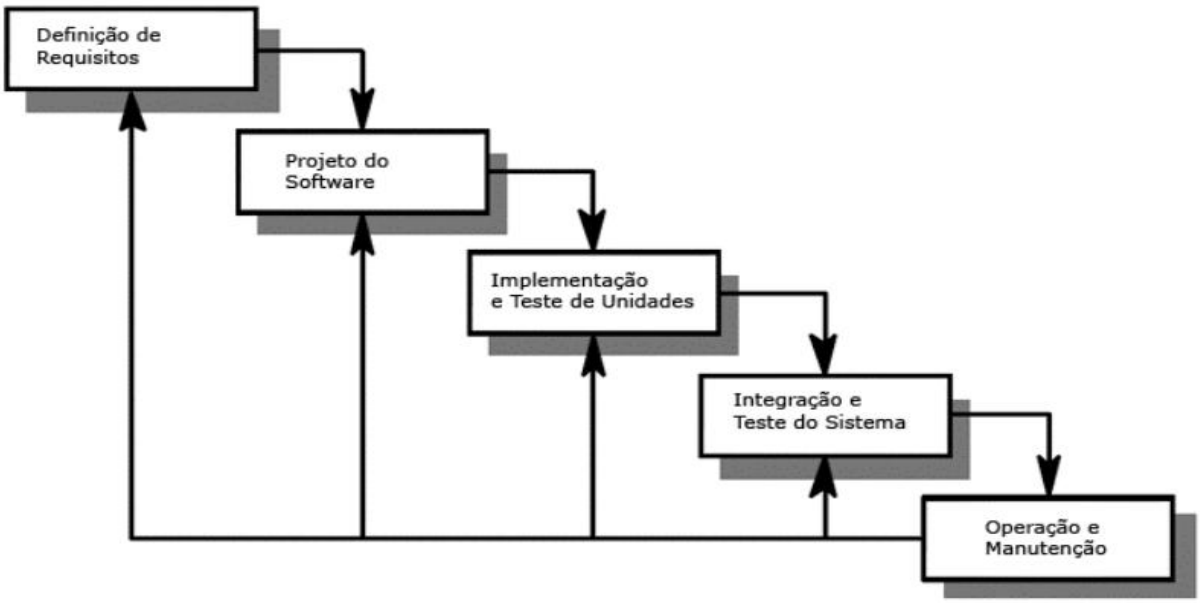


Figura 2 - Modelo em Cascata
Fonte: Sommerville, 2007, p. 44

2.2.2.2. Desenvolvimento evolucionário

O modelo evolucionário utiliza de um processo diferente do modelo em cascata, o desenvolvimento baseia-se na ideia de uma aplicação inicial, e essa aplicação inicial é refinada até o final do processo, segundo Sommerville (2007), nesse modelo as atividades de especificação, desenvolvimento e validação, são intercaladas. A Figura 3 representa isso.

Os dois tipos de fundamentais desse modelo são:

1. **Desenvolvimento exploratório**: Tem como objetivo trabalhar com o cliente a fim de explorar os requisitos e entregar um sistema final. O desenvolvimento se inicia com as partes compreendidas e no seu decorrer, são implentadas novas solicitações do cliente.
2. **Prototipação *throwaway*:** Tem como objetivo entender os requisitos do cliente e, a partir dai, desenvolver melhor definição de requisitos para o sistema. O protótipo tem como fundamento os requisitos mal compreendidos do cliente.

Sommerville (2007) considera a abordagem evolucionária mais eficaz do que a abordagem em cascata na produção de software, sendo que a vantagem se dá no fato dos requisitos serem incrementais. Porém do ponto de vista da engenharia esse modelo tem dois problemas: 1) Não produzem documentos, uma vez que em sistemas produzidos rapidamente, se torna custoso produzir documentos a cada nova versão do sistema. 2)Os sistemas acabam sendo mal estruturados, devido à mudança continua.

Esse modelo geralmente é indicado para pequenas empresas, segundo Sommerville (2007), até 500 mil linhas de código, a partir disso pode se tornar difíceis quando mais de uma equipe trabalha no desenvolvimento, nesse caso é recomendado um modelo que seja misto, aborde o modelo em cascata em conjunto com o modelo evolucionário.

Outro benefício deste modelo é o fato da validação ser intercalada com o desenvolvimento, o que tornam as necessidades de mudança e eventuais problemas mais rápidos de serem identificados, e podem ser corrigidos em tempo de desenvolvimento.

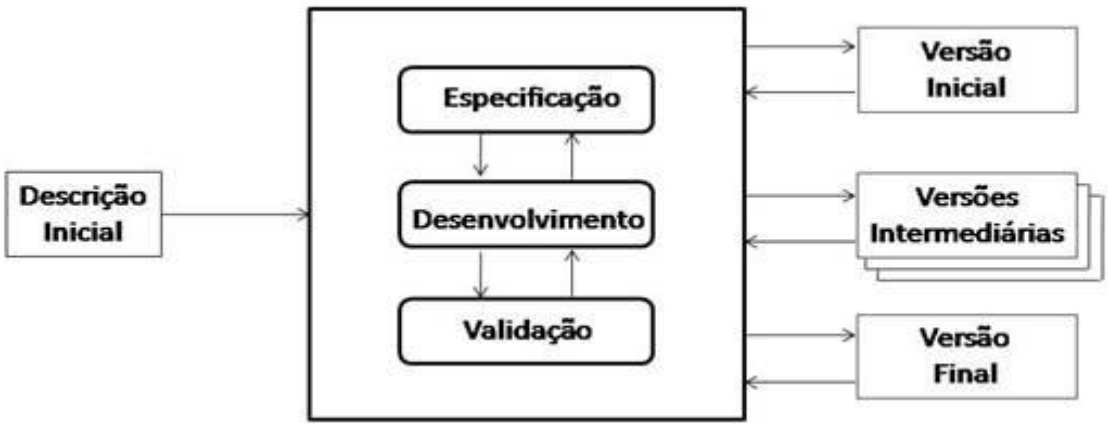


Figura 3 - Desenvolvimento Evolucionário
Fonte: Sommerville, 2007, p. 46

2.2.2.3. Engenharia baseada em Componentes

Esse modelo de software é baseado em reuso; essa abordagem depende de uma grande base de componentes de software reusáveis e algum framework de integração desses componentes. Embora os processos de requisitos e o estágio de validação sejam semelhantes a outros processos, os estágios intermediários são diferentes. A Figura 4 - Engenharia de software baseada em componentes demonstra essa diferença.

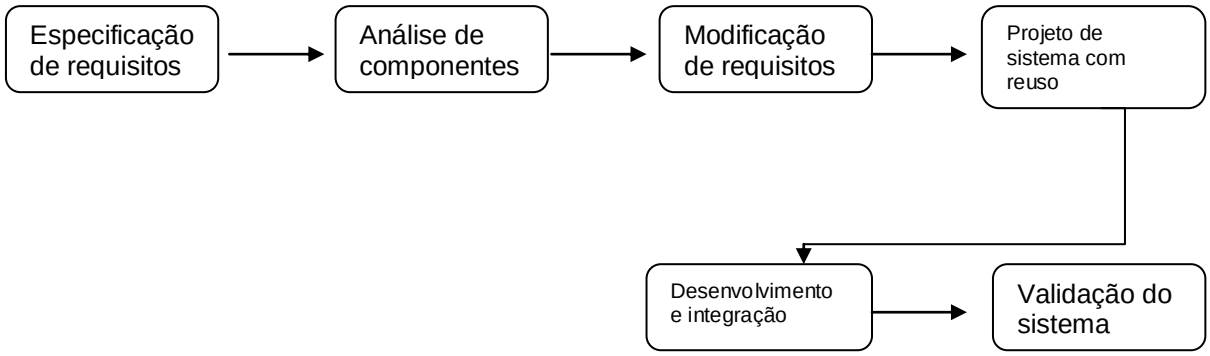


Figura 4 - Engenharia de software baseada em componentes
Fonte: Sommerville, 2007, p. 46

O modelo de engenharia baseado em componentes, foi implementado a partir do desenvolvimento baseado em componentes, que tem a filosofia, "crie uma vez, use onde quiser". Assim é possível desenvolver sistemas utilizando componentes já implementados, não sendo preciso, reescrever uma nova aplicação, codificando novamente, basta utilizar um componente já implementado anteriormente.

Porém, é possível observar que mesmo sendo esse modelo um novo conceito de desenvolvimento, a fase de validação do sistema continua inalterada, ou seja, parte do principio que os testes de software são necessários independente da forma utilizada para desenvolvimento da aplicação.

2.2.2.4. Modelo espiral

Segundo Pressman (1995) o modelo espiral, foi criado com o objetivo de abranger as melhores características do ciclo de vida clássico de desenvolvimento de software com o da prototipação. Sommerville (2007), diz que o modelo, representa o processo de software como uma sequência de atividades, com algum retorno entre uma atividade e outra. O processo neste modelo é representado como um espiral onde cada parte do espiral representa uma fase de desenvolvimento. Observe a Figura 5 - Modelo em Espiral .

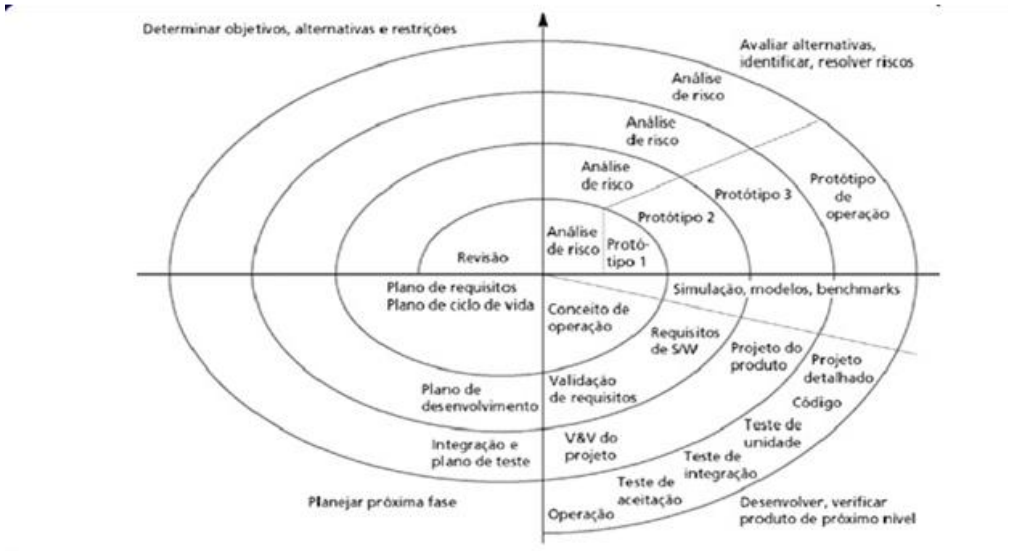


Figura 5 - Modelo em Espiral
Fonte: Sommerville, 2007, p. 49

Esse modelo tem um diferencial quando comparado aos modelos vistos anteriormente, ele leva em consideração a análise de risco, que não estava presente nos outros modelos. Segundo Pressman (1995), destaca-se por quatro atividades significativas:

> 1. Planejamento: determinação dos objetivos, alternativas e restrições.
> 2. Análise dos riscos: análise de alternativas e identificação/resolução de problemas.
> 3. Engenharia: desenvolvimento do produto no “nível seguinte”.
> 4. Avaliação feita pelo cliente: avaliação dos resultados da engenharia. (PRESSMAN, 1995, p. 38-39).

Nesse modelo a fase de validação e desenvolvimento, se encontra dentro do mesmo ciclo, e envolve teste de unidade, teste de integração e teste de aceitação, sendo um dos modelos, do ponto de vista da qualidade, que mais se aproxima do ideal. Segundo Almeida (2004),

> A adaptação de paradigmas da Engenharia de Softwares como o modelo Espiral para orientação do processo de design aparece como alternativa a uma necessidade de ampliação da qualidade dos projetos quanto ao atendimento a expectativas dos usuários e à ampliação das possibilidades de interação. (ALMEIDA, 2004, p. 469-469).

Outro ponto de vista defendido por Pressman (1995), é o fato desse modelo ser um dos modelos que mais se aproximam da realidade das empresas atualmente. “O paradigma do modelo espiral para a engenharia de software atualmente é a abordagem mais realística para o desenvolvimento de sistemas e de software em grande escala.” (PRESSMAN, 1995, p. 40).

2.3. Fundamentos de testes de software

Neste subcapitulo serão abordados conceitos de teste de software importantes para compreensão deste trabalho.

2.3.1. Objetivos do teste

Conforme descrito no capitulo anterior, no ciclo de vida de um software, dentre as etapas descritas pela Engenharia de Software, tem se a penúltima etapa, como uma das mais importantes em uma linha de desenvolvimento, o teste. Nessa etapa, o analista de teste constrói roteiros de testes que servirão para validar os módulos e correções desenvolvidas, Segundo a IEE610. 12 *apud* (GOMES, MENDES, 2013)[4] Teste é a execução de um sistema ou componente, por meios automáticos ou manuais, para verificar se ele atende a sua especificação, ou para identificar as diferenças entre os resultados obtidos e os esperados.

O teste, portanto se caracteriza como uma forma de análise dinâmica em que a implementação de um sistema ou componente é realizada de maneira a observar um conjunto de entradas e os resultados obtidos na saída, comparando se os valores dessa saída é o esperado.

Existem diversas definições de teste de software, para Hetzel *apud* Nobiato (2009): “Teste é uma atividade direcionada para avaliar um atributo ou capacidade

[4] Disponível em http://www.gotest.biz/ebts2011/apresentacoes/Experiencia_Automacao_Testes_Homologacao_PAF-CF.pdf. acessado em 22 out. 2013>

de um programa ou sistema e determinar se ele satisfaz os resultados requeridos.”, outra definição conforme Myers *apud* Nobiato (2009): “Teste é o processo de executar um programa com a intenção de encontrar defeitos” e ainda “Teste é o processo pelo qual se explora e se entende o estado dos benefícios e riscos associados com a versão de um sistema de software”.

Os testes também são utilizados para se obter métricas de requisitos não-funcionais do software, como confiabilidade ou desempenho. A realização de testes é importante para complementar outras formas de verificação e validação, pois é a única que permite exercitar o comportamento operacional do sistema.

Tendo em vista que o principal objetivo do teste de software é detectar a presença de falhas, pode se concluir que o teste é uma das fases mais importantes no processo de desenvolvimento de software.

2.3.2. Fases de teste.

O teste de software pode assumir diversas fases, preparadas durante o desenvolvimento. De maneira geral o que se sugere é um processo de desenvolvimento em “V”, conforme ilustrado na Figura 6 Neste modelo as fases de testes e desenvolvimento caminham juntas, os testes são planejados e preparados desde o inicio do desenvolvimento do software, e as atividades caminham integradas as atividades de desenvolvimento.

Normalmente o primeiro teste a ser feito no sistema é o teste de unidade, que tem o objetivo de verificar se cada módulo ou unidade do software satisfaz a sua especificação, estabelecida no projeto detalhado. Após testar cada módulo separadamente, estes módulos são agrupados para compor os subsistemas, conforme a arquitetura do software definida no projeto, e então se faz necessário o teste de integração, que visa revelar as falhas de interação entre os módulos e subsistemas.

Uma vez que os subsistemas são testados isoladamente, estes são integrados formando o software completo. A partir dai, começa a fase de testes de sistema, com o objetivo de testar o sistema como um todo, adicionando todos os componentes, tanto hardware como software, com finalidade de verificar se o sistema atende os requisitos funcionais, ou seja, sua especificação.

Após essa fase, os testes de aceitação são incorporados ao processo, com o intuito de validar o software, ou seja, determinar se ele satisfaz aos requisitos especificados na fase de especificação do sistema. Normalmente são testes realizados por clientes, conhecido também como homologação.

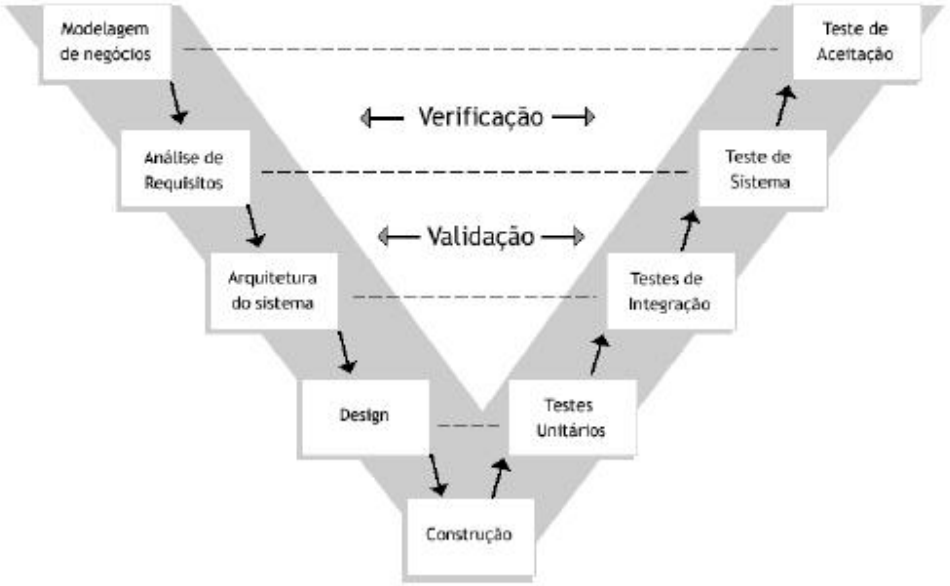


Figura 6 - Processo de desenvolvimento em V.
Fonte: Unicamp 2012, pag. 38

Os testes de regressão constituem um tipo especial de testes e visam assegurar que as alterações em partes do sistema não afetaram partes que não foram alteradas. Esses testes são aplicados principalmente em fase de manutenção.

2.4. Atividades de teste de software

O processo de testes compõe uma série de macro atividades, conforme ilustrado na Figura 7.

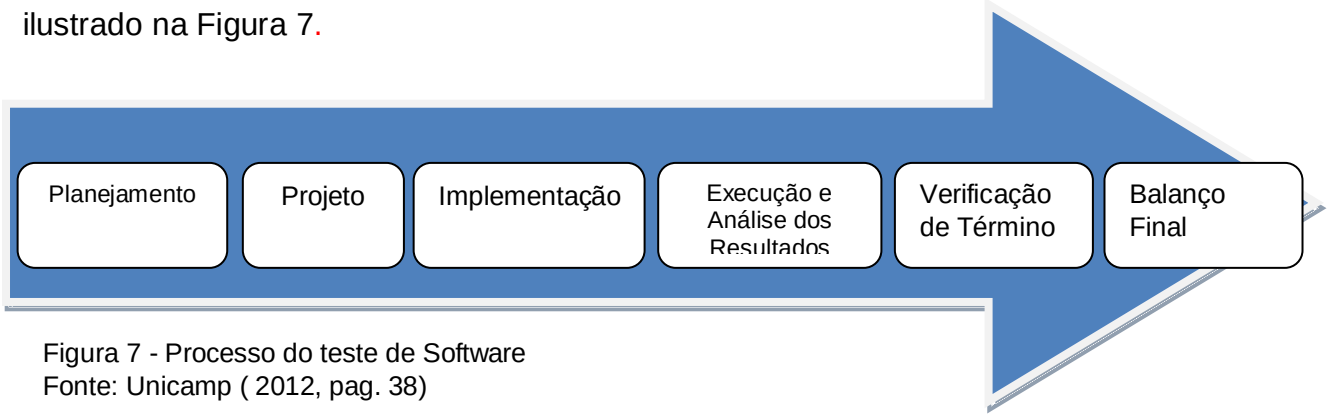


Figura 7 - Processo do teste de Software
Fonte: Unicamp (2012, pag. 38)

Planejamento: Nessa fase é definido o plano de teste a ser seguido. Segundo Sommerville (2007) , o planejamento de teste estabelece padrões de teste que não se resume apenas em descrever planos de testes, mas alocar recursos e estabelecer cronogramas. A apostila de especialização em engenharia de software da Unicamp (MARTINS, 2012) define o plano de teste como um agrupamento de sete procedimentos, sendo:

> I- Identificação das áreas de risco que devem ser testadas
> II- Determinação das fontes a serem utilizadas para a elaboração dos casos de teste.
> III - Determinação dos itens a testar
> IV - Estabelecimento das condições de completeza dos testes para cada item a testar.
> V - Especificação das condições de término dos testes
> VI – Determinação dos recursos necessários
> VII – Determinação do cronograma de teste.
> Martins (2012, p. 39)

Projeto: Nessa atividade são criados os casos de teste, os dados de entrada e o resultado esperado de saída. Os casos de teste são montados a partir da especificação do projeto, embora na ausência da documentação de requisitos ou artefatos gerados do projeto, o testador pode recorrer à solução técnica ou ao código.

O resultado dos casos de testes, é uma descrição detalhada das funcionalidades que deverão ser testadas e dos cenários possíveis para cobrir os erros gerados. Além da sequência de ações a serem executadas para realizar cada caso.

Quando os testes são realizados de forma automatizada, os procedimentos e casos de teste podem se específicos em uma linguagem, de acordo com a ferramenta utilizada para a automação de teste.

Implementação: Nesta atividade são configurados os ambientes de teste como atualização das bases de testes, configuração de servidores, instalação da aplicação a ser testada, os recursos necessários, como versão do Java, legados que serão integrados entre outros. No caso dos testes automáticos, são desenvolvidos na notação especifica da ferramenta de teste.

Execução e Análise dos resultados: Nessa atividade os casos de testes são executados e os resultados observados com o valor de saída, para então apontar os

possíveis defeitos que o software pode estar apresentando. As saídas são registradas manualmente e documentadas através de uma ferramenta como Project ou armazenados através de relatórios de log, no caso de testes automáticos. Nessa fase os defeitos também são reportados aos desenvolvedores e dependendo do modelo de software seguido pela empresa, volta ao fluxo de anterior.

Verificação e término: Segundo a Martins (2012)**,** essa atividade determina se os testes foram completamente satisfeitos e se os resultados da avaliação da qualidade foram suficientes para garantir um software livre de defeitos, caso não seja, pode ser necessários testes complementares.

Balanço Final: Nessa atividade é feita uma avaliação geral dos resultados obtidos, e os relatórios de qualidade do produto, qualidade dos testes, quantidades de erros encontrados entre outros, são emitidos.

2.5. Tipos de teste de software

Na projeção do teste de software, existem dois diferentes caminhos que podem ser optados por executar, teste por caixa branca ou por caixa preta. Independente da escolha, o resultado esperado é exatamente o mesmo, um software sem falhas.

2.5.1. Teste de caixa branca

Na projeção do teste de software por caixa branca, o testador pode ter algumas informações de quais componentes de software serão executados durante o processo de testes e quais os tipos de informações que serão manipuladas pelo mesmo, assim podendo derivar os casos de teste que:

> “Garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez; exercitem todas as decisões lógicas para valores falsos ou verdadeiros; executem todos os laços em suas fronteiras e dentro de seus limites operacionais e exercitem as estruturas de dados internas para garantir a sua validade.” (PRESSMAN, 2005, p. 793).

Testes executados pelo método da caixa branca possuem maior probabilidade de revelar *bugs*, pois o desenvolvedor ou a própria pessoa de teste sabe de antemão todos os possíveis caminhos percorridos pelo módulo. Uma boa analogia é comparar esse teste a uma caixa transparente, onde é possível ver seu conteúdo interno, assim conseguindo testar de uma forma mais eficiente.

2.5.2. **Teste de caixa preta**

Os métodos utilizando caixa preta derivam casos de testes que visam encontrar erros nas seguintes categorias como descreve PRESSMAN (2005): *"[...] funções incorretas ou ausentes; erros de interface; erros nas estruturas de dados ou no acesso a banco de dados externos; erros de desempenho e erros de inicialização e término*".

Ao contrário do teste de caixa branca, o de caixa preta é executado ao final do desenvolvimento do módulo, validando o domínio da informação, regras de negócio, valores de entrada sensíveis para o sistema, índices e volumes de dados suportados. A analogia para o teste de caixa preta deve ser interpretada de acordo com o módulo desenvolvido, por exemplo, quando o módulo desenvolvido é responsável por dizer se o numero de entrada é positivo ou negativo, mas não é possível saber qual o critério entregado para essa validação, logo de acordo com a entrada a ser testada, sabe se qual será sua respectiva saída dada como correta, qualquer coisa diferente da saída esperada, é considerado errôneo para o módulo desenvolvido.

2.5.3. Teste unitário

Após o desenvolvimento de cada unidade de software, seja ela um procedimento, função, método ou classe, é realizado o teste de unidade ou teste unitário, que visa identificar defeitos introduzidos nos algoritmos e estruturas de dados dessas unidades. Em geral, o teste de unidade é feito pelo próprio desenvolvedor da unidade.

2.5.4. Teste integrado

Durante o ciclo iterativo de desenvolvimento, cabem às pessoas responsáveis pelo desenvolvimento ou teste juntar componentes que compõem uma mesma funcionalidade para que haja o teste integrado.

De acordo com Nobiato (2009, p.11), “As unidades são então incrementalmente integradas e testadas”. Neste momento todos os testes unitários devem ter sidos executados, garantindo que todos os módulos estejam funcionando corretamente, antes de se executar os testes integrados.

Normalmente os desenvolvedores realizam testes integrados, apenas quando existe fragmentação de uma funcionalidade em módulos pequenos e plausíveis para o desenvolvimento compartilhado entre os recursos alocados, ou seja, da funcionalidade como um todo, são separados blocos que são considerados independentes para o desenvolvimento e, ao completar todos esses blocos, voltam a se reunir para o teste integrado.

Na figura Figura 8 - Fragmentação da funcionalidade em módulos entregáveistem um exemplo de como esse processo é realizado, no cenário tem a funcionalidade SPED que dentro dela, tem os módulos de contabilidade, fiscal, financeiro e comercial. Esses submódulos por consequente são designados a pessoas diferentes. O produto final dessas pessoas no final do desenvolvimento é um módulo entregável da funcionalidade SPED.

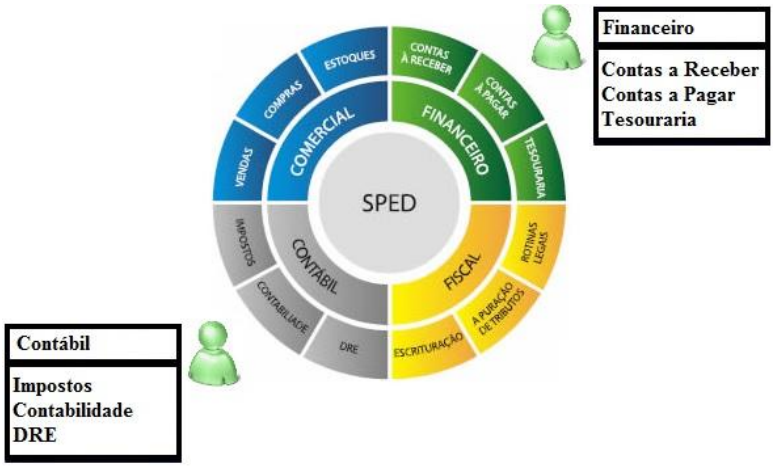


Figura 8 - Fragmentação da funcionalidade em módulos entregáveis.
Fonte: Matera (2013)

No teste desses módulos integrados o objetivo a ser alcançado é o objetivo esperado pela funcionalidade. No caso da funcionalidade descrita nessa parte do trabalho, seria um arquivo texto com as informações que envolvem as áreas de contabilidade, fiscal, financeiro e comercial.

2.5.5. Teste sistemático

Testes sistemáticos acontecem para validação do software como um todo, juntando componentes de manipulação de dados que componham aquela determinada funcionalidade a ser testada como, por exemplo, telas, relatórios, *storage procedures, triggers,* tabelas de banco de dados entre outros.

> "O teste de sistema é o nível de teste cujos requisitos são derivados da especificação de requisitos funcionais e não funcionais, e é aplicado para verificar se o software e o hardware executam corretamente ou não quando integrados ao ambiente de operação. O teste de aceitação é então conduzido para estabelecer se o sistema satisfaz ou não os critérios de aceitação definidos com o cliente." (NOBIATO, 2009, p. 11).

2.5.6. Testes de regressão

Em desenvolvimento de software a alteração de um determinado componente pode alterar o funcionamento de outro componente, causando defeitos no software não previstos anteriormente. Quando isso acontece, é possível dizer que o sistema sofreu uma regressão e se comparar que uma determinada funcionalidade, deixou de funcionar por correção de outra. Os autores Emerson Rios e Trayahú Moreira(2006, pg 15) definem que testes de regressão.

> "Visam garantir que o software permaneça intacto depois de novos testes serem realizados. Um conjunto de dados e scripts deve ser mantido como "baseline" e executado para verificar que mudanças introduzidas posteriormente não danificaram códigos já considerados bons e aceitos. Os

> resultados esperados a partir do “baseline” deve ser comparados aos resultados após as mudanças. As discrepâncias devem ser resolvidas antes de atingir o próximo nível de testes.”
>
> (Emerson Rios e Trayahú Moreira, 2006, pg 15).

Normalmente, devido aos prazos sempre apertados e o fato do teste de software ser o último processo antes do cliente, nem sempre os testadores tem tempo hábil para realizar testes de regressão. Alguma empresas tem adotado ferramentas de automação de testes para realizar esses testes.

3. TESTES AUTOMÁTICOS

Uma das alternativas utilizadas por equipes de testes de software é a automação de partes, ou total de um sistema ou funcionalidade. Uma boa pratica se considerarmos que muitas vezes o cotidiano de uma equipe de programação e testes é muito corrido e que às vezes não sobra tempo suficiente para executar outros casos de testes (teste de regressão), além dos previstos na fase atual do projeto. Na 4ª edição da revista *Testing Experience* (2008, pg. 16), o autor Alberto Vivenzio (2013), diz que a automação de teste é um tema que não pode ser ignorado e que ela faz parte do pacote de soluções de qualidade da empresa e traz melhorias de custo e eficiência. Ainda segundo o autor, automação significa a conclusão de processos funcionais recorrentes.

Em um ambiente de software, correspondem a ferramentas que são destinados a realizar essa implementação e então esse processo de teste automatizado pode ser executado. Os resultados, independente se foram executados com sucesso ou não, são então documentados em um protocolo de execução. Em geral o objetivo é que a execução do software seja realizada sem a interação com um humano.

Ainda, segundo um artigo escrito por Fabrício Ferrari (FERRARI, 2013) à automação de teste trás benefícios como: diminuição do uso de mão de obra, diminuição dos custos, aumento na velocidade do processo de teste de software, maior sustentabilidade da garantia da qualidade, perante o “triângulo da gerência de projeto” (escopo, tempo e dinheiro).

Uma justificativa muito boa para desenvolver um teste automático é quando o código da funcionalidade em questão é complexo e sofre modificações regulares. Nestes casos, o teste automático é útil, pois em modificações futuras da funcionalidade bastará rodar a build[5] para ver se a modificação feita não modificou outros cenários da funcionalidade.

Porém nem sempre a automação de teste é o canal adequado para um determinado teste, ou pode ser utilizada como substituição ao teste manual. De

acordo, com Ferrari (2013), decidir quando e como um teste deve ser automatizado exige que primeiro, se deve ter um processo de Teste de Software bem estruturado e uma equipe preparada, pois automatizar uma funcionalidade, muitas vezes exige um alto conhecimento técnico, principalmente para alguns tipos de testes específicos e é um esforço que precisa ser apoiado por uma equipe que tenha um processo de teste maduro.

Caroline (2010) escreveu alguns fatores sobre quando a automação deve ser considerada.

> 1. Frequência da execução: é importante levar em consideração a quantidade de vezes que se pretende executar os testes, se for apenas uma vez, então a execução manual pode ser suficiente.
> 2. Geração de código reusável: se o código de teste criado para um caso de teste poderá ser facilmente reutilizado em outro caso de teste, então este pode ser um bom motivo para usar alguma ferramenta de automação.
> 3. Relevância do teste: se uma funcionalidade será utilizada mais vezes do que outras, às vezes pode valer a pena criar casos de teste automáticos para ela, por exemplo, casos de teste para login na aplicação.
> 4. Esforço para automatizar: deve-se ter em mente se valerá a pena o esforço para automatizar um roteiro de teste considerando a quantidade de vezes que aquele roteiro poderá ser executado e se há casos de teste reusáveis.
> 5. Ferramentas de automação: para cada tipo de sistema a ser testado poderá ser utilizado diferentes ferramentas de automação dos testes. Isto deve ser cuidadosamente analisado antes de decidir qual ferramenta será utilizada.
> 6. Dificuldade de executar o teste manualmente: às vezes alguns casos de teste devem ser executados de forma exaustiva para um conjunto de diferentes usuários, neste caso é inviável realizar o mesmo teste para vários usuários. Logo, a automação será necessária.
> (Caroline, Blog dos Testadores, 2010).

Uma vez, decidido pela automação, o cenário mais comum é a automação do sistema, utilizando uma determinada ferramenta para fazer teste de regressão. Porém, as ferramentas para a automação de testes são diversas e podem ser encontradas em grandes quantidades. A seguir serão apresentadas algumas delas para alguns tipos de validações.

3.1. Ferramentas para testes de desempenho.

Com as ferramentas para testes de desempenho é possível validar se o sistema suporta simultâneos acessos, por exemplo, em aplicações web, ou no caso de uma migração de sistema, a transferência de uma massa de dados para o novo

[5] Versão compilada de um sistema ou uma funcionalidade integrada.

sistema. Muitas pessoas podem não conhecer esses tipos de ferramentas para automação, porém para esses tipos de situações, pode se utilizar a ferramenta *JMeter* que oferece recursos para criação de testes longos.

> "A aplicação Apache JMeter ™ para desktop é um software opensource, uma aplicação 100% Java, projetado para gerar comportamento funcional de teste e medida de desempenho. Ele foi originalmente projetado para testar aplicações web, mas desde então se expandiu para outras funções de teste."
>
> (The Apache Software Foundation, 2013[6])

Ferramenta como tal, oferece ao final da execução, um *script* funcional e parametrizável para execuções futuras que envolvem a mesma funcionalidade, ou como exemplificadas nesse caso, a execução em diversas máquinas para uma mesma funcionalidade alvo, garantindo a integridade dos dados em acessos simultâneos entre outros quesitos.

3.2. Ferramentas para testes de *backend*

Ferramentas que envolvem testes de *backend* são mais fáceis de serem implementadas, geralmente encontradas em formas de *plugin*, como é o caso do *TestNg*, que se baseia na plataforma Eclipse para rodar os testes automáticos.

De acordo com o site oficial da ferramenta TestNg (2013) [7], "O *TestNg* é um *framework* de teste inspirado nas ferramentas *JUnit* e *NUnit*, baseado na linguagem de programação JAVA e pode ser facilmente implementado". "A ferramenta deixa ao usuário alguns facilitadores, como por exemplo, a geração de relatórios dos testes executados e a que estado o mesmo se encontra (Failed, Passed ou Skipped)".

A figura 9 instalada no eclipse, apresenta o *plugin* já instalado no eclipse e logo em seguida na figura 10 a visualização do relatório que a ferramenta possibilita gerar.

---

[6] Disponível em <http://www.apache.org> Acessado em 05 Jun. 2013
[7] Disponível em < http://testng.org/doc/index.html > Acessado em 05 Jun. 2013

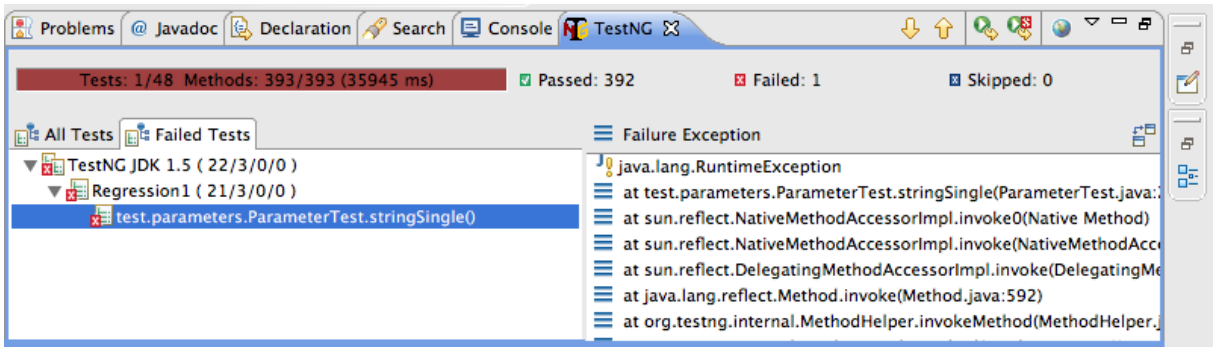


Figura 9 - Ferramenta TestNg instalada no eclipse.

Fonte: Próprio autor

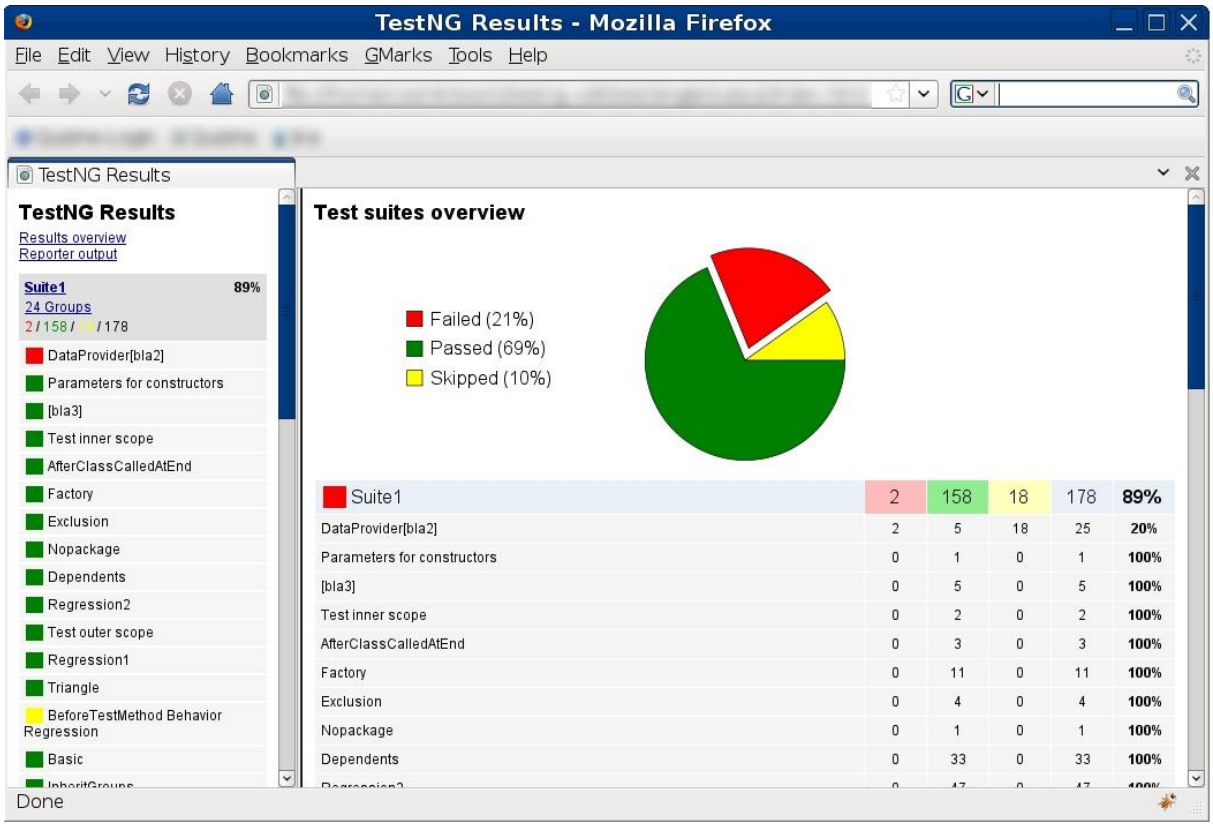


Figura 10 - Relatório gerado pela ferramenta TestNg

Fonte: Próprio Autor

3.3. Ferramentas para testes de *frontend*

Aplicar grandes mudanças e ou inserir um novo sistema em um ambiente corporativo é uma grande responsabilidade, pois envolvem processos de migração de dados e até mesmo processos para inicialização da massa de dados dentro do sistema. Ao longo desse trabalho foi dito que existe um processo paralelo ao desenvolvimento, chamado de teste, e é justamente nele que é validado se aquilo que foi feito está de acordo com o que deveria ser de fato realizado.

Além das ferramentas existentes para desempenho e as que testam processos que envolvem objetos de banco de dados, existem também as ferramentas que se baseiam em *record* e *playback,* que são responsáveis por testes, em objetos de tela, componentes como botões, mensagens de erro, campos entre outros. Com ferramentas como essas, executa se em forma de passos literalmente gravados, o que a ferramenta deverá testar durante a execução da automação, ou seja, qualquer coisa não gravada gera-se um erro.

De acordo com Kaner *apud* DevMedia (2011): “[...] os testes automatizados interagem diretamente com a interface gráfica da aplicação simulando um usuário”. Tendo essa interação diretamente com componentes de tela, as vantagens obtidas podem ser inúmeras, dentro delas, conforme Kaner *apud* DevMedia (2011), “Não requer modificação na aplicação para criar os testes automatizados. Também não é necessário tornar a aplicação mais fácil de testar por que os testes se baseiam na mesma interface utilizada pelos usuários”.

Esse tipo de teste que envolve objetos de tela, é bastante útil para empresas que sofrem migração de linguagem, neste caso é possível à validação do comportamento nativo da linguagem na qual foi desenvolvida, por exemplo, uma determinada empresa desenvolve se um produto de software em uma linguagem de programação antiga, como Centura por exemplo. A empresa que desenvolve esse produto não se adaptará rapidamente a uma transformação no processo de desenvolvimento de software, então a solução empregada seria a transformação da linguagem antiga em uma mais nova, segura e tenha profissionais já qualificados no mercado, java por exemplo. Nesse momento todos os comportamentos nativos da linguagem antiga foram literalmente transformados em comportamentos não padrões da linguagem nova. Nesse caso as ferramentas *play* e *record* conseguem

validar tais padrões de migrações, garantindo que seja equivalente aos comportamentos da antiga linguagem. A desvantagem de acordo com a DevMedia apud Kaner (2011) se dá pelo fato pequenas modificações, interferirem no processo de teste. Por exemplo: Uma simples alteração do nome de um componente de Componente01, para compenete1, se esse componente é chamado muitas vezes pode se ter um log de erro com mais problemas que na verdade tem. Entre outros pontos.

> “Existem uma forte dependência de estabilidade da interface gráfica. Se a interface mudar, os testes falham. Baixo desempenho para testes automatizados que exigem centenas de milhares de repetições, testes de funcionalidades que realizam cálculos complexos, integração entre sistemas diferentes e assim por diante”. (DevMedia *apud* KANER 2011)

Na figura 11, a tela da ferramenta Qf-Test da empresa *Quality First Software GmbH*, uma das ferramentas que fazem uso do recurso de *record* e *playback* para automação de testes baseados em interface gráfica.

Ferramentas como Qf-Test exige a compra de uma licença para uso.

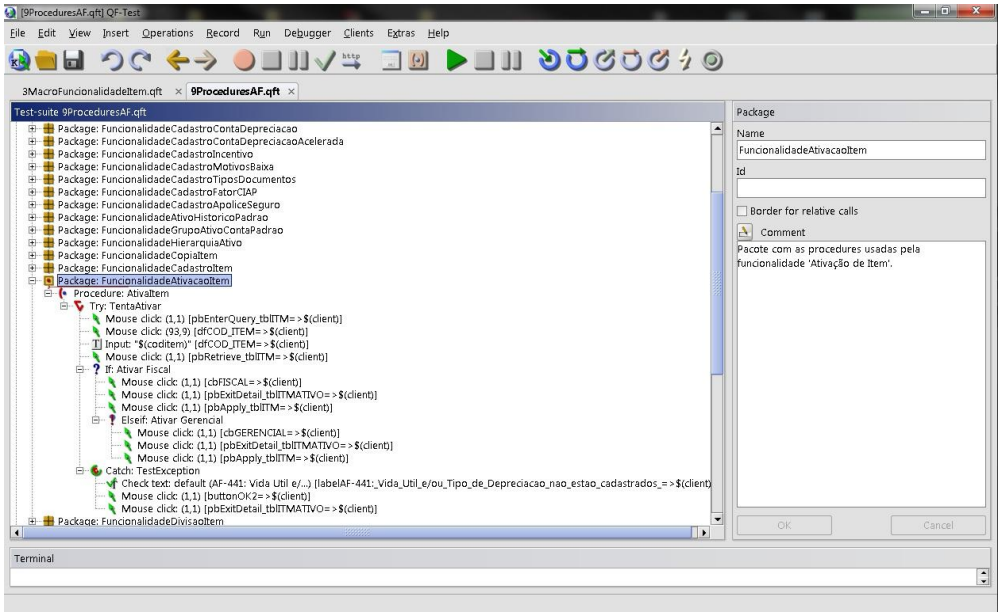


Figura 11 - Ilustração da ferramenta de automação de teste.

Fonte: Próprio autor

Entre as ferramentas que não exigem a compra de uma licença, tem se a ferramenta *Selenium*. Funciona, também, com base em *record* e *playback* e pode ser facilmente instalada através de *plugin* para *browser’s.* Observe a

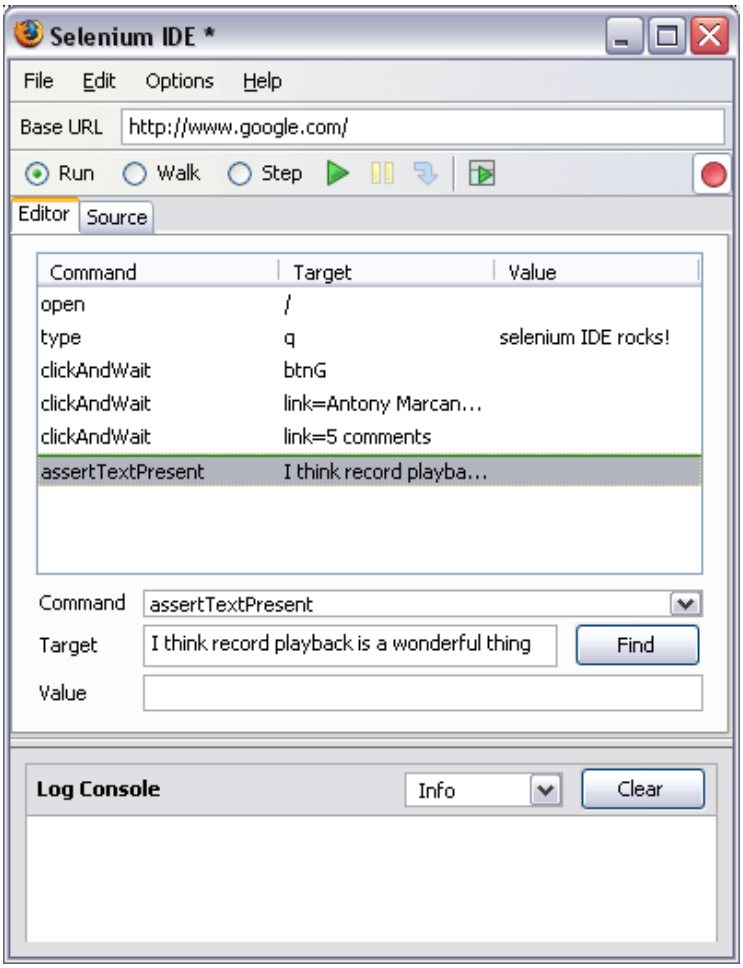


Figura 12 - Automação sendo feita pela ferramenta Selenium
Fonte: Próprio autor

Ferramentas para automação garantem que funcionalidades grandes ou de grande complexidade de regras de negócio sejam validadas em questão de minutos.

Conforme o software evolui, mudanças no código são necessárias, possíveis *bugs* podem ser introduzidos nessa etapa, no entanto essas mudanças podem ser sutis ou geradas em partes do código não previstos, portanto o número de defeitos inseridos a cada alteração pode ser extremamente grandes e possíveis de falhas no processo de teste. A Figura 12 - Relatório de erros mostra uma estimativa de erros encontrados por execução automática de um teste.

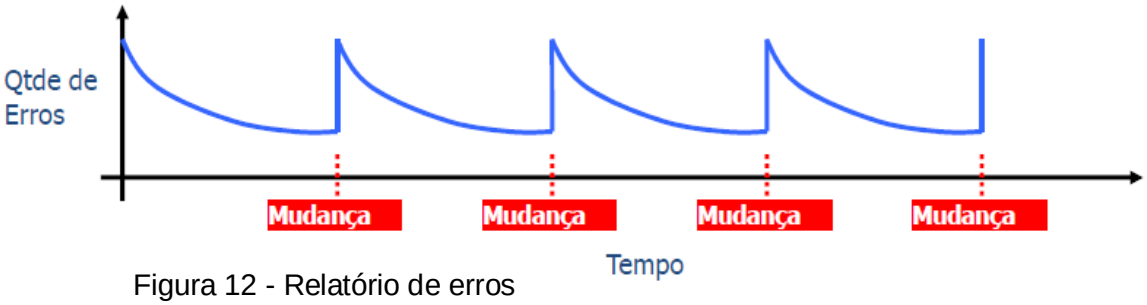


Figura 12 - Relatório de erros

Fonte: Próprio autor

Existem inúmeras outras ferramentas que podem ser utilizados para a automação de teste, neste trabalho apresenta se algumas dessas ferramentas com o intuito de demonstrar ao leitor o funcionamento da automação e qual o seu papel dentro do processo de teste.

É importante esclarecer que a automação é um facilitador de testes de regressão, mas não substitui o processo de teste manual. Algumas ferramentas de back-end como mostradas acima podem servir de facilitador de teste unitário, porém ainda é necessário que o desenvolvedor realize o teste unitário dentro do processo tradicional de desenvolvimento. A automação deve servir como um adicional ao processo de teste manual, mas o substitui.

4. BENEFÍCIOS DO TESTE DE SOFTWARE

Durante os capítulos anteriores foi feita uma abordagem sobre o teste de software e como eles podem ser utilizados dentro do processo de desenvolvimento de software, mas a pergunta inicial deste trabalho era por que testar? Quanto compensa para uma empresa investir em uma equipe de teste funcional dedicada apenas em abrir a aplicação e procurar por erros que essa aplicação possa ter, antes que isso chegue ao cliente?

Mesmo sendo o teste funcional considerado uma etapa inclusa no processo de desenvolvimento de software, muitas empresas ainda tendem a não incluir de forma efetiva estes testes em seu processo. Com o argumento de que uma equipe pode custar muito para a empresa. De fato se avaliar o valor desse profissional no mercado, realmente o custo pode ser alto dependendo do tamanho da empresa e do quanto de recurso de desenvolvimento ela tem.

Em uma pesquisa publicada pela revista Abril em 2012, Abril (2012), um Analista de Teste Junior em SP recebe em média R$ 2240,00, se acrescentar os impostos, esse profissional custará R$ 3424,71(CALCULADOR, 2012) para o empregador. Em 2006 a *testing expert* apud Caetano (2006) cita em um artigo que o número ideal de testadores em relação ao número de desenvolvedores de um projeto, é de 1 testador a cada 3 desenvolvedores, se considerar que o processo é bastante maduro e os requisitos são detalhados e consistentes. Em um cálculo simples, uma empresa que possui 21 desenvolvedores precisa ter 7 testadores, o que significa um gasto aproximado de R$ 24000 mensais.

Olhando por esta ótica manter uma equipe de teste não é muito vantajoso para uma empresa uma vez que não existe lucro direto, nas funções desempenhadas por esse profissional. Porém ao se analisar os riscos que uma empresa corre ao liberar software com defeito esses cálculos podem ser ainda maiores.

Para tal finalidade, foi criado um cenário hipotético de uma empresa chamada TCC que tem 21 desenvolvedores. Considerando que cada desenvolvedor cometa apenas um erro por desenvolvimento, ainda que seja um erro simples, o fluxo que esse percorre pode ser alarmante. Veja no cenário da Figura 13 - Fluxo de correção de defeito

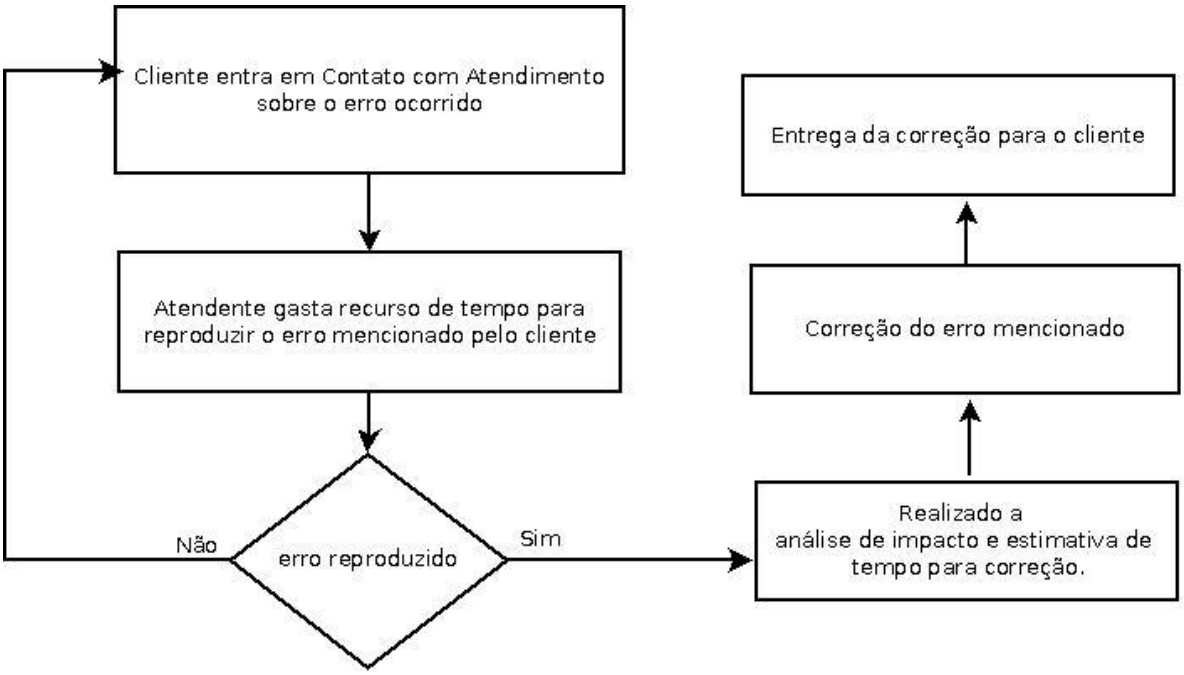


Figura 13 - Fluxo de correção de defeito

Fonte: Próprio Autor

Após esses erros serem encontrados pelo cliente, será aberto um chamado para identificar o problema desse software, se considerar apenas 01 atendente, e hipoteticamente considerar que esse atendente levara 01 hora para identificar o erro, e depois enviar esses erros ao arquiteto que fará a análise do tempo que levará a correção desse erro, para então liberar para um desenvolvedor reescrever o código. Se a equipe tiver apenas um arquiteto e considerar 01 hora para a Análise de cada erro, serão 21 horas desse arquiteto. E se forem estimadas apenas 01 hora para corrigir o erro, serão 21 horas de desenvolvimento. Mais às 21 horas do atendente.

No cálculo dos custos desse erro para a empresa, serão tomadas como base as referências usadas no cálculo dos custos de uma equipe de teste. Segundo a revista Info da Abril o salário base de um atendente Junior é de R$ 1200, de um arquiteto Junior R$ 4.816,00, e de um desenvolvedor Java Junior R$ 3000, isso custará para a empresa um total mensal de R$ 1834, R$ 7363,13, R$ 4586,67

respectivamente. Realizando um calculo simples da transformação desse salário em horas gastas por cada funcionário, basta dividir os salários mensais por 144 horas.

Observe:

Atendente: (1834/144) * 21= R$ 267,45

Arquiteto: (7363,13/144) *21 = R$ 1073,78

Desenvolvedor: (4586,67 /144) *21 = R$ 668,88

A empresa gastará para corrigir esses 21 erros um total de R$ 2010

Para um indicador mais coerente da quantidade de erros que uma empresa encontra em seu processo de teste a empresa Matera Systems forneceu seu indicador de erros encontrados por sua equipe de teste de um Módulo dos seus sistemas. Na versão 7.01.04 do Sistema de Gestão Contábil a equipe de teste encontrou 54 erros (MATERA SYSTEMS, 2013), equivalentes a aproximadamente 01 mês de trabalho, em uma equipe que conta com 5 desenvolvedores.

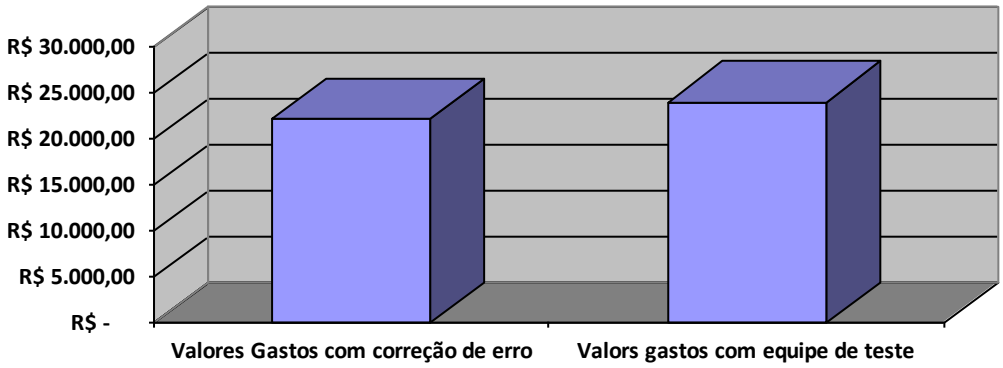
Considerando que cada desenvolvedor da empresa TCC produziu a mesma quantidade de erro e dividir 54/5 tem se um valor aproximado de 11 erros por desenvolvedor. Com essa amostra é possível validar com mais coerência o gasto que a empresa TCC gastaria com correções de erro. Se para corrigir um erro simples cometido por cada um de seus 21 desenvolvedores a empresa TCC gastava R$ 2010 reais, para corrigir 11 passará a gastar o equivalente a R$ 22110,00.

Observe que esses cálculos foram feitos considerando erros de gravidade baixa e tempo de correção não superior a 01 hora de trabalho. Obviamente que os erros cometidos em fase de desenvolvimento nem sempre são de característica simples como os simulados nesse trabalho acadêmico. Muitos são de gravidade alta e o tempo para correção é muito maior que o mencionado aqui.

O gráfico apresentado na Figura 14 - Gráfico comparativo de valores, representa a comparação entre os gastos com a equipe de teste e os gastos para corrigir os erros.

Observe que a Empresa TCC gastará aproximadamente o mesmo valor para manter uma equipe de teste, considerando os erros com classificação simples. Porém, o autor Emerson Rios (2006) diz em seu livro Documentação de Teste de Software**,** que mais de 90% dos sistemas são liberados com graves defeitos. E

afirma que “Softwares com problemas de desempenho e com defeitos de execução são custosos”. Ainda Segundo os autores em uma pesquisa realizada pelo “The Economic Impact Inadequade Infraestuture for Software Testing”, o custo total dos softwares com defeitos para as organizações nos Estados Unidos da América podem custar um pouco abaixo de 1% do produto interno bruto (RIOS, 2006). Valores muito superiores ao calculado neste trabalho.



Fonte: Próprio autor

Figura 14 - Gráfico comparativo de valores

Além disso, encontrar um erro no cliente pode custar à boa reputação para a empresa distribuidora de software e comprometer a qualidade do que está sendo entregue, o que pode gerar multas e até quebra de contrato.

No livro qualidade de software escrito por André Koscianski (2006), tem uma pequena lista sobre os maiores problemas que afetam a qualidade de um software.

> *Cronogramas não observados*
> *Projetos com tantas dificuldades que são abandonados*
> *Módulos que não operam corretamente quando combinados*
> *Programas que não fazem exatamente o que era esperado*
> *Programas tão difíceis de usar que são descartados*
> *Programas que simplesmente param de funcionar*
> (KOSCIANSKI, pag. 22, 2006).

Com exceção dos dois primeiros problemas, os demais podem ser facilmente eliminados através dos testes funcionais.

Um testador ao simular o ambiente do cliente antes da homologação, consegue perceber defeitos como os citados nessa lista, e dar sugestões de melhoria da interface para que o usuário não tenha tantas dificuldades para manipulá-lo.

Os defeitos de desempenho e não funcionamento adequado do sistema pode ser percebido através de um teste de caixa preta, ou um teste de regressão através da automação de teste. Além disso, uma vez que o testador tenha o escopo do desenvolvimento é possível validar se o software faz exatamente o que era esperado ao realizar o levantamento de requisitos. De um modo resumido, o testador valida e verifica o software desenvolvido antes de ir para o cliente.

Em suma, Filipe Giacomin, escreveu um artigo para a Testing Expert, em 2006, intitulado **Teste de Software- Para que testar?** E apresenta cinco benefícios que podem ser alcançados com teste de Software:

> ***Qualidade***
> A qualidade está relacionada ao fato de seu produto atender, ou não, as necessidades de seu cliente, sejam elas implícitas ou explícitas. Os testes ajudam a garantir que o produto atendeu todas as especificações.
> ***Economia***
> Reduz o tempo gasto com retrabalho relacionado às manutenções corretivas, muitas vezes originadas por falhas de projetos e programação.
> ***Segurança***
> Hoje, a maioria dos sistemas desenvolvidos conta com algum tipo de sistema de segurança, seja para uma área restrita de um site ou para lidar com transações de informações sigilosas. Dependendo do projeto os testes de segurança podem ser considerados fundamentais, valendo de tudo para tentar “burlar” o sistema.
> ***Confiabilidade***
> Neste caso, os testes são para medir o período máximo de tempo que o software permanece funcionando sem apresentar falhas. Muitas vezes durante os testes podem ser encontradas soluções para aumentar a confiabilidade do sistema.
> ***Negócio***
> Os testes podem gerar informações importantes para a gerência de uma empresa influenciando na decisão de liberar, ou não, o sistema desenvolvido. Neste caso, a equipe deve estudar as falhas encontradas, e então criar estratégias para eliminá-las.
> (Filipe Giacomin**,** Testing Expert, 2006).

Encontrar os erros ainda em casa é um beneficio incomparável para as empresas. Uma prova disso é o aumento por profissionais certificados na área, que vem crescendo ao longo dos anos. Uma noticia publicada por Rafael Passos, Passos (2008), já demonstrava essa preocupação por qualidade por parte das empresas fabricantes de software:

> O mercado de trabalho tem dado sinais de que cada vez mais está à procura de uma profissão relativamente nova: os testadores de software. Especialistas em encontrar falhas no desenvolvimento de sistemas, estes profissionais têm ajudado a trazer maior celeridade no processo produtivo das empresas, à medida que suas funções dispensam retrabalhos de correção em sistemas de informação.
>
> (PASSOS, 2008)[8].

O Instituto Brasileiro de Qualidade em Teste de software (IBQTST[9]) possui atualmente 136 profissionais certificados, segundo a página oficial da Instituição, acessada em 30 out. 2013. Já A Associação Latino Americana de Teste de Software (ALATS) possui 426 profissionais já certificados no Brasil, de acordo com a página oficial da Instituição, acessada em 30 out. 2013. E também a International Software Testing Qualifications Board (ISTQB[10]), possui 298.357 profissionais certificados no Mundo e 3025 no Brasil, de acordo com a página da instituição acessada em 30 out. 2013

Obviamente, um profissional certificado em teste de software tem uma visão de qualidade, e um senso crítico muito maior que um desenvolvedor e portando o teste de funcionalidade acaba sendo muito mais eficaz que um teste unitário, pensando na ineficiência dos desenvolvedores de encontrar erros em seus próprios desenvolvimentos.

De acordo com tal cenário fica evidente que os benefícios do teste mencionado nesse capitulo podem ser observados diante desse número de profissionais certificados na área. O teste de software não trás lucro direto para a intuição que o mantém em seu processo, porém o lucro indireto com ganho com qualidade, redução dos custos com defeitos, confiabilidade da marca, redução do stress com cliente causado por bugs, prevenção de multas e rescisões de contratos, entre outros, fazem com que as empresas optem por aumentar seu recurso de teste.

---

[8] Disponível em <http://itweb.com.br/voce-informa/cresce-procura-por-profissionais-certificados-em-teste-de-software/>. Acessado em 22 Out. 2013

[9] IBQTST. Instituto Brasileiro de Qualidade em Teste de software. Disponível em < http://ibqts.com.br/conteudo/show/id/15 > Acessado em 30 0ut. 2013.

[10] ISTQB International Software Testing Qualifications Board. Disponível em < http://www.istqb.org/ >acessado em 30 0ut. 2013.

5. CONSIDERAÇÕES FINAIS

O teste de software funcional, dentro do contexto de desenvolvimento que temos hoje é mais do que apenas uma fase inserida dentro do processo de software, significa garantir que o software contém defeitos ao se estressar o sistema como um todo, componentes, hardware, funcionalidade, entre outras.

Testes de caixa preta, regressão, funcionalidade evitam que a empresa encontre determinados problemas como, insatisfação do cliente, perda de credibilidade, imagem corrompida, entre outros. E em alguns casos esses danos podem ser irreparáveis ou muito difíceis de serem reparados como a perda de um contrato e credibilidade.

Esse trabalho demonstrou os benefícios que o teste de software trás as empresas de desenvolvimento uma vez que está diretamente associado à qualidade de software, prevenir que software com defeitos chegue até os clientes é uma preocupação não só do ponto de vista da qualidade, mas também da engenharia de software.

Conforme a pesquisa realizada os custos com correção de defeitos superam os gastos com testes de software. Corrigir um defeito pode gerar uma cadeia de problemas para a empresa, uma vez que a correção de um defeito pode gerar outro defeito não previsto. Neste contexto, testar o software ainda em casa, minimiza o impacto causado por um defeito, pois a própria equipe de desenvolvimento pode avaliar os problemas de falhas que esse defeito causaria no cliente e se adiantar para evitar esse problema. Além do fato que uma equipe de testes funcionais serem também responsável pela automação de sistemas completos, recurso muito importante para encontrar erros gerados em componentes inseridos por correções de defeitos.

Embora o teste manual seja predominante entre as equipes de teste, o mais coerente é ter a maior parte de todo o desenvolvimento automatizado, assim a cada desenvolvimento de novas funcionalidades, manutenções ou melhorias, esse script

de automação seria incrementado. Com esses scripts rodando em um ambiente separado, se torna possível uma métrica de qualidade comprovada, uma vez que qualquer alteração pode quebrar as builds do resultado esperado. Esse cenário é possível de ser implementado uma vez que o processo de desenvolvimento funcione adequadamente, e é possível garantir ao cliente, que é realizado um teste de regressão constante. Já que realizar testes de regressão manualmente demanda muito tempo.

No entanto, em alguns casos ainda que a companhia decida por adotar uma equipe de teste funcional no seu processo, este é realizado de forma errada, normalmente o testador não tem tempo suficiente para a execução das tarefas, e tão pouco materiais de apoio como diagramas e documentos de visão, que o ajudem a entender as funcionalidades que deverão ser testadas. Outro erro comum é confiar apenas no senso crítico do testador, quando na verdade para que o testador possa desempenhar um bom teste é necessário um trabalho em conjunto, a qualidade do software depende de que cada etapa seja realizada com sucesso. A análise de requisitos deve gerar artefatos de software que definam o escopo do projeto, a análise técnica deve ser clara e objetiva, para que não aumente o esforço de teste ao serem necessários desenvolvimentos além dos planejados, é necessária uma estimativa coerente do que precisa ser desenvolvido e testado, e o desenvolvedor deve realizar seus testes unitários garantindo que a aplicação está executando no mínimo o caminho feliz. Assim o testador não perde recurso de tempo retornando para análise de erro, defeitos simples, e pode se dedicar a encontrar problemas de desempenho, segurança, funcionalidade, entre outros.

Uma continuação possível desse trabalho seria uma pesquisa de como esse processo pode ser melhorado sem que aumente exorbitantemente o prazo de entrega e não prejudique o número de horas de desenvolvimento x horas de teste. Uma vez que foi demonstrado que existem benefícios reais de se testar software, pode se gerar uma pesquisa de qual a melhor metodologia será necessária, ou quais metodologias serão necessárias para vencer o desafio de ter uma equipe de teste funcional, que atendam as proporções e qualidade requeridas, como as mencionadas neste trabalho. E que consiga vencer barreiras como prazos apertados, escopos mal definidos, e uma cultura que infelizmente ainda tende a pensar que teste é a etapa menos importante dentro do processo de

desenvolvimento de software. Com essa pesquisa seria possível, analisar os riscos de um projeto não atender o prazo, e qual a melhor solução para desenvolver e testar em tempo hábil. Uma sugestão seria avaliar o uso de metologias ágeis, como scrum e XP, e qual a melhor maneira de integra-los de forma que a equipe de teste funcional consiga, testar e automatizar.

Contudo, o avanço da área dos últimos anos é motivador, e com um trabalho bem realizado por testadores em empresas de todo o Brasil, poderá evidenciar de forma clara e objetiva para o mercado de desenvolvimento de software os benefícios gerados ao se realizar testes de software.

REFERÊNCIAS BIBLIOGRÁFICAS:

CAETANO, Cristiano. **Proporção de testadores em relação ao número de desenvolvedores de um projeto**. Disponível em <http://www.testexpert.com.br/?q= node/399 > acessado em 25 out. 2013.

CALCULADOR**, Calculador de custo de funcionário para empresa,** Disponível em **<**http://www.calculador.com.br > Acessado em 25 out. 2013

CAROLINE, Anee. **Blog dos testadores.** Disponível em <http://gtsw.blogspot.com.br /2010/11/como-decidir-se-os-testes-devem-ser.html> acessado em 14 out. 2013.

DEVMEDIA, **Verificação, validação e testes.** Disponível em <http://www.dev media.com.br/artigo-engenharia-de-software-4-verificacao-validacao-e-testes /9879 >. Acessado em 30. ago. 2013.

FERRACI, Fabricio. **Quando automatizar?** Disponível em<http://dftestes.gershon.info/ Capitulo8.html >. Acessado em 14 out. 2013

Galeote, **Como justificar os custos associados aos testes de software**? Disponível em <http://www.galeote.com.br/blog/2010/03/como-justificar-os-custos-associados-aos-testes-de-softwarehttp://www.galeote.com.br/blog/2010/03/como-justificar-os-custos-associados-aos-testes-de-software//> Acessado em 26 out. 2013.

GIACOMIN, Filipe**. Teste de Software:** mas, para que testar? Disponível em <http://www. testexpert.com.br/?q=node/1167> acessado em 27 out. 2013.

INFO, A. **Revista tabela de salários digitais**. Disponível em <http://info.abril.com.br/ carreira/ salarios/ >. Acessado em: 25 out. 2013.

O'KEEFFE. J. Dissertação de mestrado, **análise de fatores de impacto no erro de estimativa de esforço e de duração em projetos de software,** Disponível em<http://tede.pucrs.br/tde_arquivos/2/TDE-2012-05-23T183620Z-3889/Publico/438587.pdf >. Acessado em 09 out. 2013

KOSCIANSKI. A. SOARES. M. S. **Qualidade de Software**. 2 ed. São Paulo: Editora Novatec, 2007.

MATERA Systems. **Testes Automáticos no Sistema de Gestão Patrimonial**. Disponível em: <http://www.matera.com/br/2013/03/testes-automaticos-no-sistema-de-gestao-patrim onial/>. Acessado em: 24 abr. 2013.

MARTINS**, E. Verificação e validação de software,** Instituto de Computação. Apostila de Especialização em Engenharia de software. 2012

Neto, Arilo. Introdução a Teste de Software. **Engenharia de software magazine**. ed 01, 2007, p.55.

NOBIATO, Adalberto; JINO, Mario; ARGOLLO, Miguel; BUENO, Paulo; BARROS, Celso. **Modelo de processo genérico de teste de Software**. 2009.

PRESSMAN, R. **Engenharia de Software**. 3. ed. São Paulo-SP: MAKRON, 1995.

RIOS, E. **Documentação de Teste de Software**. 2. ed. São Paulo-SP. 2006

SABADOTI, V. **Histórico sobre teste de software** disponível em <http://viniciu-ssabadoti.wordpress.com/2010/08/03/historico-sobre-testes-de-software>Acessado em: 10 ago. 2013.

SOMMERVILLE, I. **Engenharia de Software**. 8. ed. São Paulo-SP. 2007.

TESTNG. **TestNG eclipse plug-in.** Disponível em: <http://testng.org/doc/index.html > Acessado em: 20 maio 2013.

VIVENZIO, A. **Profitable, beneficial deployment of test automation in the area of SAP Core-Banking.** Disponível em<http://www.testingexperience.com/ issues/Testingexperience 04_08.pdf > acessado em 23 out. 2013.