ESCOLA SUPERIOR ABERTA DO BRASIL – ESAB
CURSO DE PÓS-GRADUAÇÃO LATO SENSU EM ENGENHARIA DE SISTEMAS

JOSÉ ERDEI

PLANEJAMENTO DE TESTE DE SOFTWARE

SÃO PAULO (SP)
2014

JOSÉ ERDEI

PLANEJAMENTO DE TESTE DE SOFTWARE

Monografia apresentada ao Curso de Pós-Graduação em Engenharia de Sistemas da Escola Superior Aberta do Brasil como requisito para obtenção do título de Especialista em Engenharia de Sistemas, sob orientação da Profa. Me. Cláudia Mara Amigo Lopes.

SÃO PAULO (SP)
2014

JOSÉ ERDEI

PLANEJAMENTO DE TESTE DE SOFTWARE

.

Monografia aprovada em .....de.....de 2014.

Banca Examinadora

_____

_____

_____

SÃO PAULO (SP)
2014

DEDICATÓRIA

Dedico este trabalho ao meu filho André e a minha companheira Selma.

AGRADECIMENTOS

Aos meus pais, pelo exemplo de vida que foram em todos os momentos, e que levo na lembrança e no coração.

Aos tutores, professores e funcionários da Escola Superior Aberta do Brasil pelo excelente atendimento, atenção e profissionalismo.

LISTA DE SIGLAS E ABREVIAÇÕES

CMMI - Capability Maturity Model - Modelo de Maturidade em Capacitação e Integração

ISTQB - International Software Testing Qualification Board

IEEE - Institute of Electrical and Electronics Engineers

PMI - Project Management Institute

SQA - Software Quality Assurance

SCM - Software Configuration Management- Gerenciamento de Configuração de Software

UML - Unified Modeling Language

LISTA DE ILUSTRAÇÕES

RESUMO

Esta pesquisa trata do desenvolvimento de testes durante a elaboração de softwares. A etapa de testes é um componente muito importante no processo de desenvolvimento de um software de qualidade e deve ser vista como uma fase que visa abranger toda a engenharia de software e não apenas uma tarefa final do projeto. Como objetivo, este trabalho propõe a utilização de um modelo de apoio ao planejamento de testes, com ênfase em testes funcionais, o que possibilita a prevenção e eliminação de defeitos para que o software esteja em conformidade com os requisitos pré-definidos, e atenda completamente as necessidades dos clientes. Os aspectos metodológicos avaliaram a solução proposta comparando estudos de resultados da literatura e pesquisas na internet. Foram obtidas consideráveis informações a respeito do assunto, como a qualidade de software em relação aos custos de software, tipos e técnicas de testes, caso de testes e sua relação ao caso de uso. Os resultados mostraram que as boas práticas em testes de software utilizando algumas técnicas e modelo de apoio aos testes melhoraram muito sua qualidade. Finalmente este trabalho concluiu indicando o modelo dos Planos do Plano Global do Projeto (PMBOK) como uma proposta de apoio no planejamento de desenvolvimento de software, e como os casos de testes podem ser gerados a partir dos casos de uso.

Palavras-Chave: Planejamento; Caso de Teste; Caso de Uso.

SUMÁRIO

1 INTRODUÇÃO

Nos dias atuais a necessidade de se aprimorar a qualidade de produtos de software tem incentivado um contínuo aperfeiçoamento de atividades de testes. Sem prejuízo dos demais aspectos, pode-se afirmar que a qualidade de um software está correlacionada à qualidade dos testes aos quais ele é submetido.

Dentro da atividade de testes de software, a geração de casos de teste é fundamental. É importante que os casos de teste verifiquem para um conjunto significativo de exemplos de dados de entrada se os requisitos especificados foram devidamente implementados. Tais requisitos são geralmente escritos sob a forma de casos de uso. Por essa razão diversos trabalhos têm proposto derivar casos de teste a partir de casos de uso.

Com o crescimento da tecnologia a necessidade de se aprimorar a qualidade dos produtos de software tem incentivado o aperfeiçoamento das atividades de testes. A questão que queremos destacar é como obter maior qualidade no desenvolvimento de software utilizando um modelo de apoio no planejamento de testes de software.

Este trabalho tem como objetivo geral propor um modelo de apoio ao planejamento de teste de software, com ênfase em testes funcionais e que possibilite a geração de casos de teste a partir de casos de uso.

E como objetivos específicos:

descrever os conceitos de atividades de testes de software;

descrever as principais técnicas de teste de software;

descrever os tipos de testes de software;

descrever os conceitos de caso de uso;

mostrar métodos para geração de caso de testes com utilização de casos de uso.

Este trabalho será realizado através de pesquisa exploratória, tendo como principal fonte a pesquisa bibliográfica em livros e publicações especializadas, além de artigos, trabalhos acadêmicos de dissertações e teses disponíveis para consulta na Internet.

Este trabalho está subdividido em cinco capítulos. Na primeira parte apresenta-se a Qualidade de Software. No segundo capítulo iremos acompanhar o conceito de Testes de software. No terceiro capítulo descreve-se o Planejamento de teste de

Software. No quarto capítulo apresentamos o caso de uso, seus elementos e formas de representação. No quinto capítulo apresentamos o caso de testes suas derivações e sua elaboração utilizando os casos de uso. E por fim uma conclusão, definida pelos resultados e opiniões obtidas ao decorrer do trabalho, seguida pela apresentação de propostas para trabalhos futuros.

2 QUALIDADE DE SOFTWARE

Para realizar os procedimentos de testes com qualidade, devemos considerar os objetivos mensuráveis de qualidade na qual queremos alcançar.
É normal confundir o controle de qualidade com a garantia de qualidade, muitas equipes de garantia de qualidade acabam praticando controles de qualidade, para evitar este equívoco podemos dividir estes procedimentos em dois métodos, os métodos preventivos e os métodos de detecção, assim podemos diferenciar as atividades de garantia de qualidade com as atividades de controle de qualidade.

2.1 DEFINIÇÕES DE GARANTIA E CONTROLE DE QUALIDADE

Muitas equipes de testes tem dificuldades de identificar as atividades de garantia de qualidade das atividades de controle de qualidade, para amenizar essa dificuldade descrevemos abaixo, as diversas definições de garantia e controle de qualidade.

Segundo Bastos (2012), a garantia de qualidade tem como definição ser um conjunto sistemático e um planejamento de atividades e seu principal objetivo é a melhoria contínua dos processos no desenvolvimento de software, para tanto deverá ser proposta pelos responsáveis da empresa que tenham como visão estratégica uma política de qualidade, suas atividades estão ligadas as práticas que irão garantir que durante os primeiros trabalhos na elaboração de um novo sistema, estes estarão em conformidade com seus requisitos e atenderão as necessidades dos usuários, o controle de qualidade são atividades posteriores as das atividades de garantia, quando uma não conformidade ou erro é encontrado, são tomadas as devidas providências para que o produto ou procedimento tenha o retorno ou resultado conforme os requisitos e resultados esperados.

> Tanto a garantia de qualidade quanto o controle de qualidade se distinguem da função de auditoria interna. A auditoria interna é uma atividade independente dentro da organização, tem o propósito de revisar as operações e é uma tarefa de gerenciamento. É um controle gerencial para medir e avaliar a eficiência de outros controles.(BASTOS, 2012,p71).

Na visão do trabalho a qualidade tem duas definições, uma do ponto de vista do produtor, e outra do ponto de vista do consumidor, na primeira definição a

qualidade é quando os requisitos estão dentro do que foi especificado, na segunda definição é quando atende as necessidades do cliente.

Na garantia de qualidade, o CMMI - Capability Maturity Model - Modelo de Maturidade em Capacitação e Integração, propõe um gerenciamento com visibilidade para a eficácia, produzindo assim um software de qualidade, revendo e auditando partes do produto, para verificar se estão de acordo com os padrões definidos fornece os resultados das revisões no formato de checklists predefinidos pelos responsáveis no processo de garantia de qualidade.(BASTOS, 2012).

Podemos dizer que a garantia de qualidade, segundo Bastos(2012), é uma atividade que analisa e estabelece o processo para gerar determinado produto e deve ter como atividades para: determinar a metodologia de desenvolvimento de sistemas, medições de processos, manutenções de sistemas, definição de requisitos e processos e padrões te teste.

Depois de definir esses processos, a garantia de qualidade deve medi-los a fim de identificar as fraquezas e depois corrigi-las para a melhoria contínua do processo de qualidade.

O controle de qualidade se concentra nas atividades de identificação de defeitos em produtos e começam no início do processo de desenvolvimento do software, com a revisão dos requisitos, e continuam até que o teste do aplicativo esteja completo e o sistema esteja implantado.

Segundo explica Bastos(2012), pode haver controle de qualidade, sem controle de qualidade, uma equipe de teste pode conduzir vários testes no final do desenvolvimento de um sistemas, não havendo a metodologia de desenvolvimento focada na qualidade deste o início do desenvolvimento com atividades ou medidas para garantir a qualidade do produto com mais facilidade de detectar erros ou defeitos já no início do seu desenvolvimento.

As afirmações de Bastos(2012), a seguir ajudam a diferenciar controle de qualidade e garantia de qualidade:

O controle de qualidade:

- está relacionado a um produto ou serviço específico;
- verifica se um produto ou serviço específico tem um atributo específico;
- identifica defeitos com o propósito principal de corrigi-los;
- é responsabilidade da equipe/do funcionário;
- é responsabilidade da equipe/do funcionário.

A garantia de qualidade:

- ajuda a estabelecer processos;
- determina programas de medida para avaliar processos;
- identifica fraquezas em processos e os aperfeiçoa;
- é uma responsabilidade de gerenciamento;
- está relacionada com todos os produtos que serão gerados por um processo;
- avalia se o controle de qualidade está funcionando.

A equipe que trabalha na garantia de qualidade não deve estar envolvida nas atividades de controle de qualidade, a não ser para validá-las.
Hoje em dia é considerado m sistema de software de qualidade, quando atinge níveis adequados de confiabilidade na realização de suas funcionalidades; para isso, deve atender às categorias da qualidade da ISO/IEC 9126-1.
Atualmente, além da garantia das funcionalidades, é necessário atingir metas de desempenho, principalmente em plataformas aplicadas na Web.
O teste, como parte do processo de desenvolvimento de software, é um dos meios de aumentar a garantia de confiabilidade e desempenho dos sistemas desenvolvidos.

2.2 CARACTERÍSTICAS DE QUALIDADE SEGUNDO A ISO 9126-1

Devemos ter uma atenção especial ao classificar os riscos do projeto de testes e os riscos da ocorrência de defeitos, como qualquer projeto, existe uma lista de possibilidades de riscos que podem fazer com que o software seja mal testado, ocorrendo vários defeitos, ou seja um software mal testado haverá maior ocorrência

de defeitos, para minimizar estas ocorrências de defeitos, devemos seguir as características de qualidade pela norma ISO 9126-1.

Como explica Bastos (2012), A NORMA ISO/IEC 9126-1 estabelece as características de qualidade que todo software deve ter, conforme listado no quadro a seguir.

Característica	Descrição
Funcionalidade	Capacidade do software de fornecer funcionalidades que atendam a necessidades definidas quando usado sob determinadas condições preestabelecidas.
Confiabilidade	Capacidade do software de manter um nível específico de desempenho quando usado sob determinadas condições preestabelecidas.
Usabilidade	Capacidade do software de ser entendido, aprendido, usado e atrativo quando empregado sob determinadas condições específicas.
Eficiência	Capacidade do software de manter o desempenho, em relação aos recursos disponíveis, quando usado sob determinadas condições específicas.
Manutenibilidade	Capacidade do software de ser mantido.
Portabilidade	Capacidade do software de ser transferido de um ambiente para outro.

Figura 1 – Algumas características de qualidade - NBR ISO/IEC 9126-1.
Fonte: Bastos (2012, p.99).

a) **Funcionalidade (Satisfação das Necessidades)**: É a característica do software que tem as funcionalidades que satisfaça as necessidades quando o software está em uso dentro das condições especificadas.

b) **Confiabilidade (Imunidade a Falhas)**: É a característica do software que mantém um nível especificado de performance quando o software está em uso dentro das condições especificadas.

c) **Usabilidade (Facilidade de Uso)**: É a característica do software de ser entendido, aprendido, usado e atrativo quando o software está em uso dentro das condições especificadas.

d) **Eficiência (Rápido e "Enxuto")**: É a característica do software que tem uma performance apropriada, relativa ao conjunto de recursos usados quando o software está em uso dentro das condições especificadas.

e) **Manutenibilidade (Facilidade de Manutenção)**: É a característica do software de ser mudado. Modificações incluem correções, melhorias ou adaptações do software de mudar em um ambiente, e em requisitos e especificações funcionais.

f) **Portabilidade (Uso em outros Ambientes)**: É a característica do software de ser transferido de um ambiente para outro.

Para garantir que o software não perca nenhuma das características de qualidade estabelecias pela norma, faz-se necessária uma associação entre os tipos de teste necessários para que uma característica de qualidade seja atendida (BASTOS, 2012).

Característica	Exemplos de testes
Funcionalidade	Teste de funcionalidade
Confiabilidade	Teste de estresse
Usabilidade	Teste de usabilidade
Eficiência	Teste de desempenho
Manutenibilidade	Teste caixa-branca etc.
Portabilidade	Teste de produção, teste alfa etc.

Figura 2 – Características de qualidade e os tipos de teste necessários.
Fonte: Bastos (2012, p.99).

2.3 CUSTOS DA QUALIDADE DO SOFTWARE

Na maioria das empresas, o teste é executado como uma etapa dentro do processo de desenvolvimento, e são executado pelos próprios desenvolvedores e pelos usuários do sistema, mas serve apenas para validar que as especificações e os requisitos do negócio foram implementados.

Como explica Bastos (2012), neste processo de teste de desenvolvimento gera produtos com defeito, sendo necessário detectar esses defeitos e corrigi-los, para isso a melhor maneira de testar um software é ter um processo de teste claramente definido.
Na maioria das vezes, os defeitos existentes nos softwares, constituem-se em riscos para o negócio e para a imagem da empresa, para minimizar os riscos causados por defeitos por desenvolvimento, a empresa deve adotar uma metodologia própria.
Os defeitos existentes nos softwares, na maioria das vezes, constituem-se em riscos tanto para o negócio quanto para a imagem da empresa. O objetivo de um processo

de teste, com metodologia própria, é minimizar os riscos causados por defeitos provenientes do processo de desenvolvimento.
A equipe de testes, nesta nova abordagem, os testes deverá ser feito por uma equipe de especialistas treinados para isso e não mais por analista de sistemas que as vezes é obrigado e até contra sua vontade de executar essa atividade.

Nem os analistas de sistemas nem os usuários são técnicos de teste de software, eles não possuem aquele perfil do técnico que vai fundo no software para buscar defeitos. Para serem feitos corretamente, é necessário que os testes sejam executados por profissionais capacitados, usando metodologia apropriada, em ambiente adequado e, muitas vezes, tendo à disposição ferramentas de automação.

No desenvolvimento de sistemas, na maioria das vezes os prazos são pequenos, previamente estabelecidos. Nem sempre há prazos específicos para os testes, eles precisam ser feitos dentro do prazo estipulado para o desenvolvimento. Numa situação crítica como essa, é muito importante o uso de uma metodologia adequada e de uma equipe treinada e capacitada. Desse modo, através de revisões e inspeções, os testes podem ser feitos nos documentos de desenvolvimento tão logo o projeto de desenvolvimento do software se inicia.

Segundo Bastos(2012), o projeto de teste de software deve começar paralelamente ao projeto de desenvolvimento. A correção de defeitos encontrados em requisitos ou modelos custa menos. Ao reduzir a incidência de defeitos encontrados nos testes propriamente ditos, estamos otimizando essa atividade. Existe também a possibilidade de priorizar as funcionalidades que trazem maiores riscos para o negócio, a equipe de teste dará prioridade àquelas funcionalidades nas quais os riscos para o negócio são maiores: se não for possível testar tudo, pelo menos testaremos o que é mais importante. E tudo isso é baseado em técnicas que serão escolhidas de acordo com a situação. Lembre-se de que testar é um exercício de reduzir os riscos que as aplicações podem trazer para os negócios.
Ao tratar os testes como um processo organizado e muitas vezes paralelo e integrado ao processo de desenvolvimento, os custos de manutenção serão reduzidos com toda a certeza.

A figura 3 mostra a Regra 10 de Myers, que estabelece que o custo de correção de defeitos tende a aumentar quanto mais tarde o defeito é detectado. Defeitos encontrados durante a produção tendem a custar muito mais que defeitos encontrados em modelos de dados e em outros documentos do projeto do software(BASTOS, 2012).

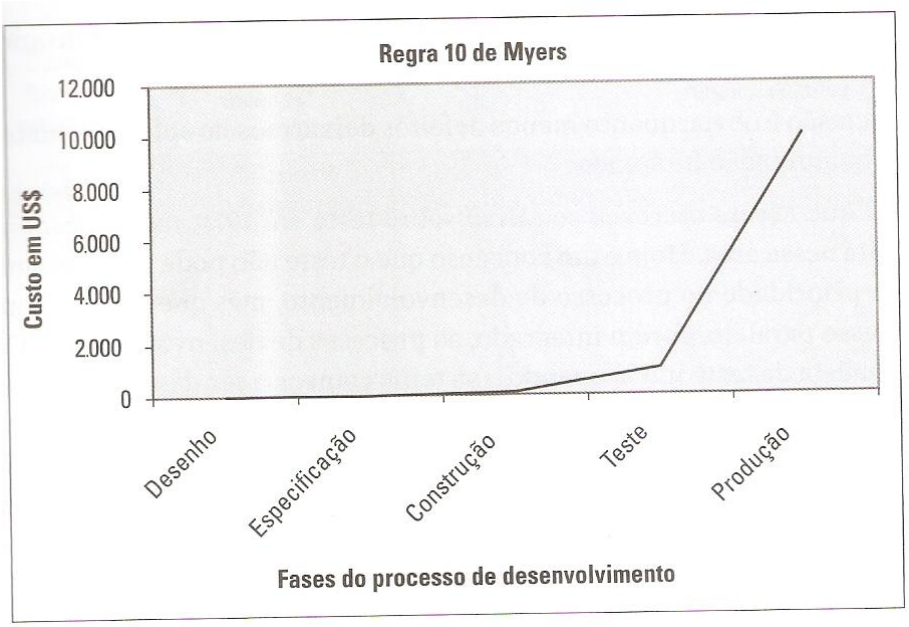

Figura 3 – Custo da correção dos defeitos.
Fonte: Bastos (2012, p.19)

Segundo Bastos(2012), em 1976, Michael Fagan publicou um artigo sobre inspeções de código no qual explicava como esse tipo de trabalho poderia reduzir os defeitos dos softwares.

Também em 1976, Glenford Myers lançou o livro Software reliability principles and practices(Nova York: Wiley), mostrando a importância de os testes serem feitos corretamente e já introduzia o conceito de casos de teste, no entanto, foi com seu livro The art of software testing(Nova York: Wiley), lançado em 1979, que ele criou os conceitos sobre teste de software, uma publicação que, na época, tornou-se uma das bíblias da qualidade de software. Nesse livro, Myers afirmava o seguinte:

- Os testes unitários podem remover entre 30% e 50% dos defeitos dos programas.
- Os teste de sistemas podem remover entre 30% e 50% dos defeitos remanescentes.

- Desse modo, os sistemas podem ir para produção ainda com aproximadamente 49% de defeitos.
- Por último, as revisões de códigos podem reduzir entre 20% e 30% desses defeitos.

Devemos Lembrar que uma aplicação que está sendo desenvolvida hoje, no futuro, poderá ainda estar rodando e gerando defeitos daqui a dez ou quinze anos, existem estatísticas sobre idades de sistemas que provam isso. E são imprevisíveis os custos da ocorrência de defeitos à medida que os anos vão passando, muitas vezes é difícil encontrar o técnico que trabalhou naquela aplicação ou até mesmo técnicos que usem a linguagem de programação empregada, pois a tecnologia muda ao longo do tempo, chegamos a conclusão que é óbvia: quanto menos defeitos deixarmos no software, mais barata será sua manutenção no futuro (BASTOS, 2012).

Segundo Bastos (2012), muita pesquisa vem sendo feita nessa área. Hoje é um consenso que o teste não pode mais ser um apêndice sem prioridade no processo de desenvolvimento, mas que precisa constituir um processo paralelo, porém integrado, ao processo de desenvolvimento. O trabalho da equipe de teste inicia quando o sistema começa a ser desenvolvido, e assim os defeitos passam a ser detectados nas primeiras fases de desenvolvimento. Isso implica um custo inicial maior de teste, devido ao investimento necessário para montar a equipe (espaço, equipamentos, pessoal, ferramentas etc.); por outro lado, há uma redução no custo da correção de defeitos. Todos os estudos mostram que, no final, sai mais barato investir em testes.

Com a execução dos testes dentro de um processo próprio (metodologia, ciclo de vida etc.), o custo de correção de defeitos na produção cai muito (Regra 10 de Myers). O custo do software (desenvolvimento + manutenção) tende a ser relativamente menor quando o software é bem testado, conforme se vê nas figuras 4 e 5.

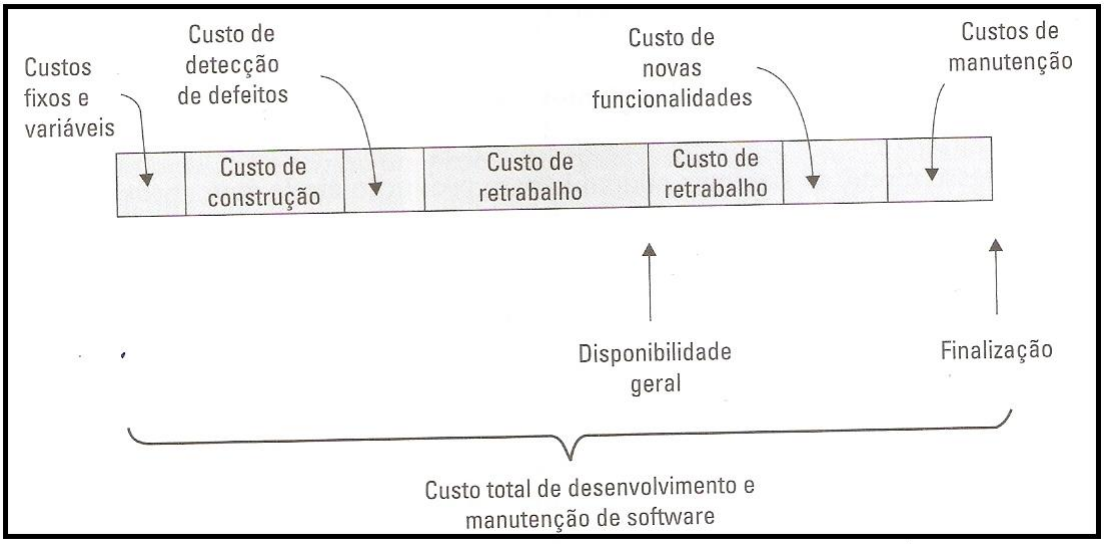

Figura 4 – Custo total de software.
Fonte: Bastos (2012, p.20).

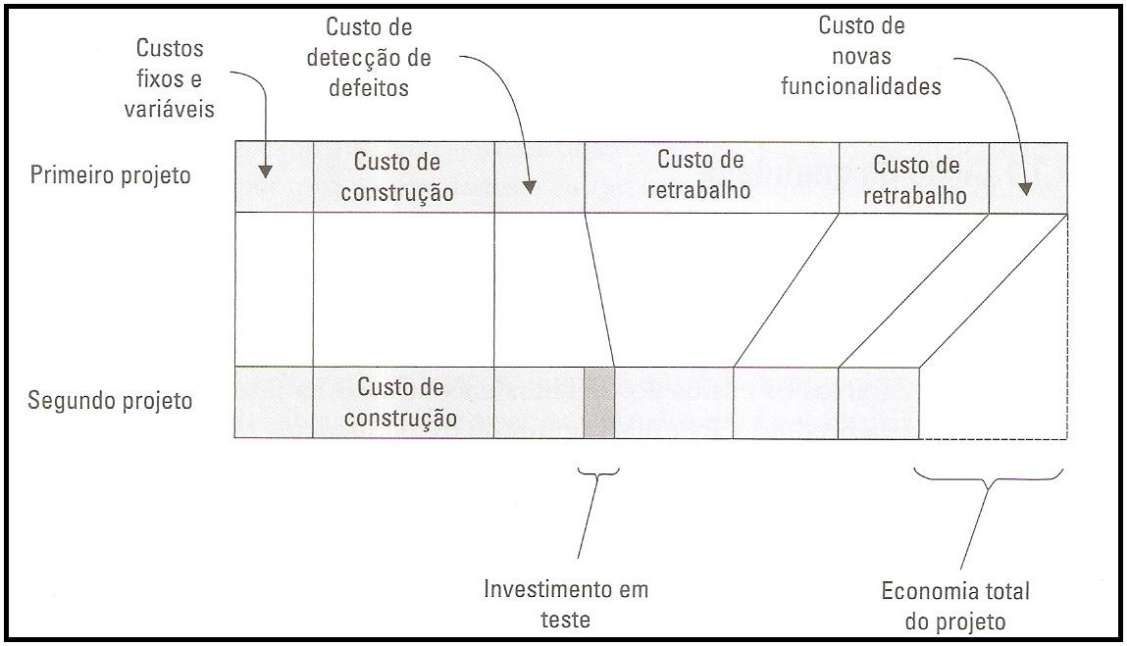

Figura 5 – Benefícios do investimento em teste de software.
Fonte: Bastos (2012, p.21).

Como vimos pela Regra 10 de Myers, o custo da correção dos defeitos tende a subir quanto mais tarde ele é corrigido. A correção de defeitos encontrados na fase de desenho custa menos que a de defeitos encontrados na produção. Alguns especialistas, como Rex Black, afirmam que existe uma progressão do tipo "dez, cem, mil" se comparamos os custos dos defeitos encontrados na fase do desenho, da codificação e da produção. Rex Black, autor do livro Managing the esting process (Redmond: Microsoft Press, 1999), uma das referências para os estudiosos de teste

de software, publicou um artigo no qual tenta mostrar mais detalhadamente essa evolução de custos (BASTOS, 2012).

Estrutura de teste	Teste informal	Teste manual	Teste automatizado
Pessoal	0	60.000	60.000
Infraestrutura	0	10.000	10.000
Ferramentas	0		12.500 (*)
Total do investimento	0	70.000	82.500
Defeitos encontrados pela equipe de desenvolvimento			
Defeitos encontrados	250	250	250
Custo dos defeitos	2.500	2.500	2.500
Defeitos encontrados pela equipe de teste			
Defeitos encontrados	0	350	500
Custo dos defeitos	0	35.000	50.000
Defeitos encontrados em produção			
Defeitos encontrados	750	400	250
Custo dos defeitos	750.000	400.000	250.000
Custo da qualidade			
Conformidade (investimento em melhorias)	0	70.000	82.500
Não conformidade (custo total dos defeitos encontrados)	752.500	437.500	302.500
Total do custo da qualidade	752.500	507.500	385.000
Retorno do investimento	–	350%	445%

Figuras 6 – Estimativas de custos de testes (valores em dólares).
Fonte: Bastos (2012, p.22.).

Como explica Bastos (2012), no artigo "The cost of software quality", Black usa como exemplo um processo de teste sem automação no qual os preços são expressos em dólares norte-americanos. Por suas estimativas, conforme se vê na figura 6, a correção dos defeitos encontrados nos testes de desenvolvimento (unitário e integração) custa dez dólares, em média. A correção dos defeitos encontrados por uma equipe de teste custaria, também em média, cem dólares.

Considerando-se todo o investimento necessário para montar a equipe para a execução dos testes no caso exemplificado, seriam gastos 106 mil dólares. No entanto, essa equipe encontraria mais 350 defeitos que deixariam de ocorrer na produção, fase em que os mesmos defeitos custariam em torno de mil dólares. Isso equivale a dizer que a empresa lucraria ao introduzir um processo de teste, mesmo que manual, uma vez que os defeitos passariam a ser encontrados nas fases iniciais do processo de desenvolvimento do software, quando repetimos, a correção custa muito menos.

Numa visão mais simples do quadro montado por Rex Black, podemos dizer que o retorno do investimento será maior à medida que investirmos em teste. Isto é, quanto mais melhorarmos a atividade de teste, melhores serão os resultados financeiros. Essa é a mensagem que nos quis passar esse conceituado pesquisador.

Evidentemente, existem alguns tipos de negócio em que a ocorrência de defeitos na produção pode trazer prejuízos muito maiores. Um carro de Fórmula 1 parou, segundo os técnicos, por um defeito no software que controlava o veículo. Essa falha, causada por uma condição inesperada, obrigou a equipe a sair de uma corrida na qual vinha tendo um bom desempenho. Nesse caso, os custos foram incalculáveis. Outro episódio, muito divulgado no Brasil, foi o de uma empresa da área de telecomunicações que deixou de emitir seus carnês de cobrança por um defeito de software. Alguns milhões de reais foram arrecadados em atraso, com um custo adicional de milhões por perda de receita. Nesses casos, o custo do defeito pode ultrapassar bastante os mil dólares previstos pelo artigo de Rex Black (BASTOS, 2012).

3 CONCEITO DE TESTES DE SOFTWARE

Segundo Bastos (2012) o teste de software é um processo de investigação para detecção de defeitos nos sistemas/processos, para fornecer informações se os mesmos se comportam conforme as especificações do projeto. Isso significa que os testes de software são uma peça fundamental para tratar os riscos existentes em um software, procurando garantir sua qualidade.

Ainda segundo Bastos (2012) o principal objetivo do processo de teste é encontrar o maior número possível de defeitos no software. Essa é a grande missão de uma equipe de teste. Os documentos básicos para definir os objetivos de teste são os requisitos do negócio e, caso tais requisitos não se encontrem disponíveis ou estejam mal definidos, os objetivos poderão ser buscados em reuniões com os usuários ou desenvolvedores. Algumas empresas criam requisitos de teste a partir dos requisitos do negócio. Os objetivos servem para ajudar a definir também o escopo do projeto de teste, atividade importante no gerenciamento de projetos.

A prática de gerar os requisitos de teste a partir dos requisitos do negócio facilita de modo considerável traçar objetivos capazes de cobrir todas as necessidades especificadas pelos usuários. Isso também permite garantir que sejam cobertos todos os requisitos que precisam ser testados. O ideal seria manter uma linha conectando cada requisito aos correspondentes casos de teste, ou seja, ao executarmos um ou mais casos de teste, já poderíamos identificar a qual requisito eles se referem (BASTOS, 2012).

Abaixo serão apresentados os tipos de testes como: Verificação e Validação, Caixa Branca e Caixa Preta, testes de Unidade, testes de Integração, testes de Sistemas e testes de Aceitação, procurando mostrar suas características, suas técnicas ou processos de aplicação.

Os testes Caixa Preta ou testes Funcionais têm como objetivo garantir que os requisitos e especificações do sistema sejam atendidos e estão divididos nos testes de Requisitos, testes de Regressão, testes de Tratamento de erros, teste de Suporte Manual, testes de interconexão com outros softwares, testes de controle e testes paralelos.

3.1 TESTES DE VERIFICAÇÃO E VALIDAÇÃO

Segundo Bastos (2012), nos testes de Verificação estão incluídos todos os testes que permitem verificar se o software está sendo construído corretamente, ou seja, teste unitário, teste de integração e teste de sistema. Na Validação estão incluídos os testes que permitem validar se o software está fazendo o que foi definido nos requisitos, ou seja, teste de aceitação.

Outra maneira de diferenciar verificação de validação, como explica Bastos (2012), consiste em responder às perguntas:

construímos corretamente o sistema?

nós construímos o sistema correto?

A primeira pergunta diz respeito ao que foi construído, e a segunda ao entendimento do que era para ser construído.

Podemos considerar como algumas das atividades de verificação:

Teste de aceitação;

Teste de performance (desde que exista uma exigência de requisito);

Teste de segurança (idem).

Como podemos ver, as atividades de verificação e validação estão distribuídas nas diversas etapas do processo de testes e, de uma maneira genérica, a verificação é executada antes da validação. Ou seja, verificamos a construção e validamos o software (BASTOS, 2012).

3.2 TESTES CAIXA BRANCA E CAIXA PRETA

O objetivo dos testes Caixa Branca, também chamado de testes estruturais, é garantir que o produto seja estruturalmente sólido e que funcione corretamente. Busca-se, assim, determinar se a tecnologia foi usada de modo adequado e se os componentes montados funcionam como uma unidade coesa (BASTOS, 2012).

As técnicas para esse tipo de teste são desenhadas não para garantir que o sistema seja funcionalmente correto, mas sim para que ele seja estruturalmente robusto.

Os testes Caixa Preta ou testes funcionais do sistema são desenhados para garantir que os requisitos e as especificações do sistema tenham sido atendidos. O processo

costuma envolver a criação das condições de teste para uso na avaliação da correção da aplicação (BASTOS, 2012).

Como este trabalho está focado apenas nos testes Caixa Preta ou funcional listamos abaixo somente as técnicas que atendem a este tipo de teste:

Testes de requisitos: Os testes de requisitos visam verificar se o sistema executa corretamente as funcionalidades e se é capaz de sustentar essa correção após sua utilização por um período de tempo contínuo. Os testes de requisitos devem ser considerados formalmente realizados após os programas se tornarem operacionais, embora os requisitos possam ser testados individualmente durante as fases anteriores do ciclo de vida (BASTOS, 2012).

Ainda segundo Bastos (2012), os objetivos dos testes de requisitos consistem em verificar se:

- os requisitos dos usuários estão implementados;
- a correção é mantida por períodos prolongados de tempo;
- o processamento da aplicação está em conformidade com as políticas e os

procedimentos da organização.

De modo complementar, verificar se foram incluídas as necessidades provenientes de outras fontes, tais como:

- responsável pela segurança;
- Administrador de Banco de Dados (DBA);
- auditores externos;
- controlador;
- procedimentos contábeis;
- regulamentos (governamentais e de outros órgãos).

As informações para todos os testes foram retiradas de Bastos (2012):

Testes de regressão: Os testes de regressão voltam a testar segmentos já testados após a implementação de uma mudança em outra parte do software.
Objetivos:

Sempre que mudanças são efetuadas num segmento de código, problemas podem ocorrer em outros segmentos já testados. Desse modo, entre os objetivos dos testes de regressão, temos:

- determinar se a documentação do sistema permanece atual;
- determinar se os dados e as condições de teste permanecem atuais;
- determinar se as funções previamente testadas continuam funcionando corretamente após a introdução de mudanças no sistema.

Testes de tratamento de erros: Os testes de tratamento de erros determinam a capacidade do sistema de tratar apropriadamente transações incorretas.
Objetivos:
Em alguns softwares, aproximadamente 50% do esforço de programação é dedicado ao tratamento das condições de erro.
Os objetivos específicos dos testes de tratamento de erro são:

- determinar se todas as condições de erro esperadas são reconhecidas pelo sistema;
- determinar se foi atribuída responsabilidade para processar os erros identificados;
- determinar se é mantido um controle razoável sobre os erros durante o processo de correção.

Testes de suporte manual: A parte manual do sistema exige tanta atenção com relação aos testes quanto à parte automatizada processada no computador. Embora os momentos e os métodos de teste possam ser diferentes nessas duas partes, os objetivos dos testes são os mesmos.
Objetivos:

- verificar se os procedimentos de suporte manual estão documentados e completos;
- determinar se as responsabilidades pelo suporte manual foram estabelecidas;
- determinar se o pessoal que dará o suporte manual está adequadamente treinado;

- determinar se o suporte manual e o segmento automatizado estão interligados apropriadamente.

Testes de interconexão com outros softwares: Os softwares de aplicação costumam estar interconectados com outros softwares de mesmo tipo.
A interconexão pode se dar através de dados recebidos, fornecidos ou de ambos. Esses testes são desenhados para garantir que a interconexão entre os softwares de aplicação funcione corretamente.
Objetivos:

- determinar se os parâmetros e dados são transferidos corretamente entre os softwares;
- garantir o momento certo de execução e a existência de coordenação das funções entre os softwares;
- determinar se a documentação pertinente é correta e completa.

Testes de controle: Entre os controles estão a validação de dados, a integridade de arquivos, as trilhas de auditoria, o backup e a recuperação, a documentação e outros aspectos do sistema relacionados à integridade. Embora os testes de controle estejam incluídos em outras técnicas de teste, as técnicas de teste de controle são desenhadas para assegurar o funcionamento dos mecanismos que supervisionam o funcionamento do sistema de aplicações.
Objetivos:
O teste de controle é a ferramenta de gestão necessária para assegurar que o processamento seja realizado conforme sua intenção. Os objetivos dos testes de controle são garantir que:

- os dados estejam completos e corretos;
- as transações sejam autorizadas;
- a manutenção das informações da trilha de auditoria seja realizada;
- os processamentos sejam eficientes, eficazes e econômicos;
- o processamento atenda às necessidades dos usuários.

Testes paralelos: O teste paralelo é uma das mais populares técnicas de teste; entretanto, com os softwares se tornando mais integrados e complexos, crescem as

dificuldades para a realização dessas técnicas, e sua utilização tende a diminuir. O teste paralelo serve para determinar se os resultados de um novo software de aplicação são consistentes com o processamento do antigo sistema ou da antiga versão do sistema.

Objetivos:

- Assegurar que a nova versão do software de aplicação execute corretamente.
- Demonstrar consistências e inconsistências entre duas versões do mesmo software de aplicação (BASTOS, 2012).

3.3 TESTES DE UNIDADE

Estágio mais baixo da escala de teste, aplicado aos menores componentes de código criados, visando garantir que eles atendam às especificações funcionais e arquiteturais. Costuma ser feito pelo programador e testa as unidades individuais: funções, objetos e componentes.

3. 4 TESTES DE INTEGRAÇÃO

Teste do sistema ao término de cada interação, dentro de um ambiente operacional controlado, para validar a exatidão e a perfeição na execução de suas funções, referentes aos casos de uso da iteração. Em geral, é realizado pelo analista de sistemas para um módulo ou conjunto de programas.

3. 5 TESTES DE SISTEMAS

Execução do sistema como um todo, dentro de um ambiente operacional controlado, para validar a exatidão e a perfeição na execução de suas funções, acompanhando cenários sistêmicos elaborados pelo profissional de requisitos do projeto. Costuma ser feito pelo analista de teste (caso de teste) em ambiente de teste.

3. 6 TESTES DE ACEITAÇÃO

É a última ação de teste antes da implantação do software, sendo sua execução de responsabilidade do cliente. O objetivo do teste de aceitação é verificar se o software está pronto e pode ser usado por usuários finais para executar as funções e tarefas para as quais foi construído. Em geral é feito pelo usuário em ambiente de homologação.

4 PLANEJAMENTO EM TESTE DE SOFTWARE

O teste de software é o processo que visa executar o software de maneira controlada, tendo como seu principal objetivo avaliar seu comportamento de acordo com o que foi especificado anteriormente, a execução dos testes é considerada um tipo de aprovação.

Segundo Bastos (2012), o primeiro passo para a execução de um teste bem feito é seu planejamento. Para fazer um bom planejamento, é necessário usar um documento padronizado, pelo menos no ambiente da empresa, mas que seja guiado por regras e normas internacionais. Não é preciso inventar um documento para realizar essa atividade.

O documento que permite elaborar o planejamento de teste é conhecido como Plano de Teste, e é nele que definimos o nível de cobertura e abordagem dos testes. O manual do QAI cita outro documento, chamado Estrutura de Teste, no qual se encontra a definição do que vai ser selecionado para teste e a descrição dos resultados esperados, embora esse mesmo manual cite que ambos os documentos podem estar juntos no Plano de Teste. De qualquer maneira, nunca é demais repetir que os testes, para ser realizados adequadamente e para atingir os objetivos esperados, precisam ser planejados (BASTOS, 2012).

O Plano de Teste deve ter como base os requisitos da aplicação e os requisitos de teste. Existem, inclusive, ferramentas de automação que ajudam na elaboração desse documento, sendo que algumas podem ser baixadas livremente na internet.

A metodologia TMap define um documento chamado Estratégia de Teste, que deve ser elaborado antes do Plano de Teste; contudo, é mais comum o Plano de Teste ser o principal e único documento de planejamento dos testes, incorporando o que o QAI chamou de Estrutura de Teste e o que a metodologia TMap define como Estratégia de Teste (JORGENSEN, 2012). Acreditamos que não seja necessário mais de um documento para definir essa atividade, mas às vezes é necessário mais de um Plano de Teste num determinado projeto.

Por que os Planos de Teste são importantes?

Segundo Bastos (2012), a partir do momento em que começamos a tratar os testes como um projeto ou processo, e não mais como uma etapa ou atividade dentro do processo de desenvolvimento precisou planejá-los. O Plano de Teste é uma maneira de documentar o projeto de teste, deste modo, permitir que os mesmos testes possam ser repetidos e controlados. Além disso, o Plano de Teste define o nível de cobertura que deverá ser alcançado no projeto de teste. Cada projeto de teste deverá seguir o ciclo de vida do processo de teste e, por conseguinte, ter pelo menos um Plano de Teste específico. Os sistemas complexos podem ter mais de um Plano de Teste. Quando, posteriormente, o software passar por algum tipo de manutenção e precisar voltar a ser testado, esse documento será um ótimo guia para orientar a repetição ou servir de base para executar os testes de regressão.

PLANEJAMENTO DO PROJETO DE TESTE

A execução de testes, visando cobrir todo o sistema e garantir que ele não terá mais nenhum defeito, é uma atividade custosa e, muitas vezes, de realização impossível. O Plano de Teste é o documento que permitirá definir o nível de cobertura segundo o qual os elementos mais críticos do software serão testados com prioridade e com um nível de cobertura mais amplo. (Como elementos críticos consideramos aqueles classificados pela análise de riscos ou assim caracterizados pelos usuários do software.) Existe um padrão mundialmente aceito que lista o conteúdo de um Plano de Teste (IEEE, 1990). O QAI segue, em linhas gerais, esse mesmo padrão, embora com algumas características próprias. Nenhum dos dois padrões, porém, fala sobre estimativas ou métricas de teste, mas muitas empresas já estão utilizando métricas para definir o tamanho de seu projeto de teste e estimar o esforço necessário para sua execução Bastos (BASTOS, 2012).

Segundo Bastos (2012), A Figura 7 estabelece uma relação entre o modelo definido pelo PMBOK e os modelos criados pela IEEE 829 e pelo próprio QAI. Em nosso entender, o Plano de Testes deveria seguir o modelo do Plano do Projeto do PMBOK. No entanto, verificamos que existe uma relação entre os três modelos.

O Plano Global do Projeto, conforme definido pelo PMBOK, e como explicado por Bastos (2012), é composto de vários planos específicos, como se vê na primeira coluna do quadro:

Escopo:

O escopo de teste tem por objetivo básico definir com precisão o que será e o que não será coberto pelo teste.

Custos:

Para estabelecer ou estimar os custos de um projeto de teste é necessário haver dentro da empresa uma métrica afinada a seu ambiente, é preciso medir o tamanho do projeto de teste para saber quanto ele vai custar.

Tempo:

A estimativa de tempo e, conseqüentemente, a elaboração do cronograma está ligada diretamente ao tamanho do projeto que, por sua vez, servirá de base para o cálculo dos custos.

Qualidade:

A medição da qualidade deve ser acompanhada através de um programa de indicadores a ser implementado no decorrer do projeto e deve estar de acordo com as necessidades de qualidade estabelecidas pelo cliente.

Integração:

O projeto de teste integra-se sobretudo com o projeto de desenvolvimento. Eventualmente poderão existir integrações com outros projetos de teste. Além disso, como é usual que o projeto seja segmentado em etapas, é preciso manter a integração entre esses elementos.

Recursos humanos:

A quantidade de homens/hora estimada para o projeto é estabelecida após a estimativa de seu tamanho, sendo necessário definir os recursos envolvidos em cada etapa do projeto.

Funções e responsabilidades:

Nesta parte do Plano de Teste, especifica-se o que cada um terá de fazer dentro do projeto de teste.

Comunicação:

As regras de gerenciamento de projetos preconizadas pelo PMI definem a necessidade de um gerenciamento de comunicação para dar suporte aos projetos com objetivo de garantir a maneira como as partes envolvidas no projeto receberão as informações de que precisam para tomar decisões.

Riscos:

O líder do projeto de teste deve tomar muito cuidado para não misturar os riscos do projeto de teste com os riscos do negócio, o que pode causar confusão e redundância caso esse trabalho tenha de ser repetido.

Suprimentos:

Pode ser que algumas ferramentas precisem ser compradas ou um ambiente de teste tenha de ser configurado ou preparado, e essas atividades poderão envolver aquisição de softwares ou equipamentos.

Conclusão:

O importante é que o planejamento dos testes seja feito com muito cuidado e precisão. O modelo do PMBOK define o que é necessário para que um projeto seja bem gerenciado. Os outros modelos são apenas decorrência desse modelo genérico, ou melhor, uma adequação específica ao projeto de teste.

Os itens dos Planos de Teste sugeridos pelo IEEE e pelo QAI estão listados nas colunas seguintes. O que procuramos aqui é estabelecer uma relação entre esses diversos planos:

Planos do Plano Global do Projeto (PMBOK)	Itens do Plano de Teste do modelo IEEE 829	Itens do Plano de Teste do modelo do QAI
Escopo	Identificação do Plano de Teste Referências Introdução Funcionalidades a serem testadas Funções a serem testadas do ponto de vista do usuário Funções que não serão testadas do ponto de vista do usuário	Escopo do teste Objetivos de teste Premissas Análise de risco Estrutura do teste Ferramentas de teste
Custo	Métricas	Métricas
Tempo	Cronograma	Estrutura do teste Recursos e cronograma de testes
Qualidade	Abordagem (estratégia) dos testes Critérios de conclusão dos testes Critérios para interrupção e retomada dos testes	Gerenciamento de dados de teste
Integração	Ambiente de teste	Ambiente de teste
Recursos humanos	Pessoal (equipe, treinamento, local etc.) Responsabilidades	Funções e responsabilidades
Comunicação	Entregas (Plano de Teste, Casos de Teste etc.)	Comunicação
Riscos	Riscos do processo de teste Plano de riscos e contingências	Análise de risco
Suprimentos	–	Ferramentas de teste

Figura 7 – NBR ISO/IEC 9126-1 - Características da Qualidade de Software.
Fonte: Bastos (2012).

5 CASO DE USO

Um Caso de Uso representa uma unidade discreta da interação entre um usuário (humano ou máquina) e o sistema. Ele representa as funcionalidades que um sistema deve prover e uma indicação que qual elemento, denominado de ator, pode acessar uma determinada funcionalidade. Um ator é um humano ou entidade máquina que interage com o sistema para executar um trabalho no contexto do sistema.

Segundo Valente (2007), um caso de uso é uma seqüência com especificação de interações entre um sistema e os agentes externos que utilizam esse sistema. Um caso de uso deve definir o uso de uma parte da funcionalidade de um sistema, sem revelar a estrutura e os comportamentos internos desse sistema. Um modelo de casos de uso típico contém vários casos de uso.

Um caso de uso representa quem faz o que com o sistema, sem considerar o comportamento interno do sistema.

Elementos de caso de uso:

Na terminologia da UML[1], qualquer elemento externo que interage com o sistema é denominado ator. Vamos detalhar melhor essa definição. O termo “externo” indica que atores não fazem parte do sistema. E o termo “interage” significa que um ator troca (envia e/ou recebe) informações com o sistema (VALENTE, 2007).

1 . A **Unified Modeling Language** (**UML**) é uma linguagem de modelagem que permite que desenvolvedores visualizem os produtos de seus trabalhos em diagramas padronizados, junto com uma notação gráfica (www.uml.com).

CATEGORIA DE ATORES	EXEMPLOS
Pessoas	Empregado, cliente, gerente, almoxarife, vendedor
Organizações	Empresa fornecedora, Agência de Turismo, Administradora de Cartões
Outros Sistemas	Sistema de Cobrança, Sistema de Estoque, Sistema de Vendas
Equipamentos	Leitora de Código de Barras, Sensor, Plotter, catraca eletrônica

Figura 8 – Categoria de Atores.
Fonte: Valente (2007).

Levando em conta que um ator é um papel representado no sistema, o nome dado a esse ator, portanto, deve lembrar o seu papel, em vez de lembrar quem o representa. Exemplos de bons nomes para atores: Cliente, Estudante, Fornecedor, etc. Exemplos de nomes inadequados para atores: João, Fornecedora ACME, etc. (VALENTE, 2007).

Tipos de relacionamentos:

A UML define vários tipos de relacionamentos no modelo de casos de uso: comunicação, inclusão, extensão e generalização. Vamos dividir em blocos para ficar mais fácil didaticamente:

INTERAÇÕES	COMUNICAÇÃO	INCLUSÃO	EXTENSÃO	GENERALIZAÇÃO
Caso de Uso e Caso de Uso		*	*	*
Ator e Ator				*
Ator e Casos de Uso	*			

Figura 9 – Interações.

Fonte: Valente (2007).

Comunicação:
Representa a interação do ator com o caso de uso, ou seja, a comunicação entre atores e casos de uso, por meio de envio e recebimento de mensagens.
A comunicação entre casos de uso são sempre binárias, ou seja, envolvem apenas dois elementos. Representam o único relacionamento possível entre atores e casos de uso. Por exemplo: O ator Correntista envia e recebe mensagens do Caso de Uso Calcular Empréstimo Pessoal, por um relacionamento de comunicação.

Generalização

Ocorre entre Atores ou Casos de Uso. É quando temos dois elementos semelhantes, mas com um deles realizando algo a mais. Também é comparado ao relacionamento de extensão, com uma variação do comportamento normal sendo descrita sem muito rigor. Segue o mesmo conceito da orientação a objetos.

Por exemplo: num Caso de Uso no qual C2 herda de C1, significa dizer que temos C1 como um Caso de Uso mais genérico e C2 mais específico. Outro exemplo: podemos criar um ator genérico **Aluno** e especializá-lo nos atores **Aluno Matriculado** e **Aluno Ouvinte**.

Extensão:

Um relacionamento extensão entre Casos de Uso indica que um deles terá seu procedimento acrescido, em um ponto de extensão, de outro Caso de Uso, identificado como base. Os pontos de extensão são rótulos que aparecem nos cenários de Caso de Uso base. Pode se colocar diversos pontos de extensão em um mesmo Caso de Uso, inclusive repetir um mesmo ponto de extensão.
Um Caso de Uso de extensão é utilizado para:
1. Descrever rotinas de exceção ou para expressar o desmembramento de um Caso de Uso.
2. Separar um comportamento obrigatório de outro opcional.

3. Separar um trecho do Caso de Uso que será executado apenas em determinadas condições.
4. Separar trechos que dependam da interação com um determinado ator.

Inclusão:

São alguns cenários cujas ações servem para mais de um Caso de Uso deve-se utilizar um relacionamento de inclusão. Mostra que um deles terá seu procedimento copiado em num local especificado no outro Caso de Uso, identificado como base (VALENTE, 2007).

Um relacionamento de inclusão na descrição de um Caso de Uso base, é representado com o termo Include seguido de seu nome. Para evitar a cópia de trechos iguais ganhamos tempo de modelagem e também de implementação, e de manutenção, ao trabalhar com Casos de Uso de inclusão.
Exemplo: Para validar a Matrícula é útil para Casos de Uso como Renovar Matrícula de Aluno, Emitir Histórico Escolar, Lançar Notas de Provas, entre outros (VALENTE, 2007).

Formato, detalhamento e abstração

Os casos de usos podem ser compreendidos conforme as descrições abaixo, as definições de Formato, Detalhamento e Abstração e a figura nº 10.
Segundo Valente (2007), quanto ao FORMATO, comumente são utilizadas a descrição contínua, a descrição numerada e a descrição particionada. Esses tipos de narrativa são apresentados a seguir, com o exemplo correspondente ao caso de uso de saque de determinada quantia em um caixa eletrônico de um sistema bancário.
No formato de descrição contínua a narrativa é feita através de um texto livre. Como exemplo considere o caso de uso **Realizar Saque** em um caixa eletrônico:

> O Cliente vai ao caixa eletrônico e insere seu cartão. O sistema pede a senha do Cliente. Após fornecer sua senha e esta ser validada, o Sistema exibe as opções de operações possíveis. O Cliente opta por realizar um saque. Então o Sistema requisita o total a ser sacado. O Sistema fornece a quantia desejada e imprime o recibo para o Cliente (VALENTE, 2007).

Ainda segundo Valente (2007) no formato de descrição numerada, a narrativa é descrita através de uma série de passos numerados. Considere como exemplo o mesmo caso de uso **Realizar Saque**:

1. Cliente insere seu cartão no caixa eletrônico.

2. Sistema apresenta solicitação de senha.

3. Cliente digita senha.

4. Sistema exibe menu de operações disponíveis.

5. Cliente indica que deseja realizar saque.

6. Sistema requisita quantia a ser sacada.

7. Cliente retira a quantia e o recibo.

O estilo de descrição particionada tenta prover alguma estrutura à descrição de casos de uso. A seqüencia de interações entre o ator e o sistema é particionada em duas colunas, uma para o ator e outra para o sistema:

CLIENTE	SISTEMA
Insere seu cartão no caixa eletrônico.	
	Apresenta solicitação de senha.
Digita senha.	
	Exibe menu de operações disponíveis.
Solicita realização de saque.	
	Requisita quantia a ser sacada.
Retira a quantia e o recibo.	

Figura 10 – Interações entre Ator e Sistema.
Fonte: Valente (2007).

O grau de detalhamento a ser utilizado na descrição de um caso de uso pode variar desde o mais sucinto até a descrição envolvendo vários detalhes (expandido) (VALENTE, 2007).

Um caso de uso resumido descreve as interações sem detalhes. Um caso de uso expandido descreve as interações em detalhes.

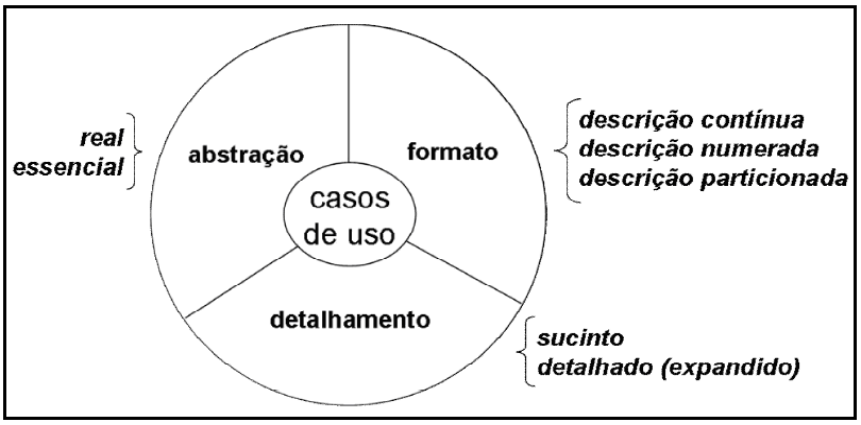

Figura 11 – Caos de Uso, Formato, Detalhamento e Abstração.
Fonte: Valente (2007).

O grau de abstração de um caso de uso diz respeito à existência ou não de menção à tecnologia a ser utilizada na descrição desse caso de uso. Em relação ao grau de abstração, um caso de uso pode ser real ou essencial (VALENTE, 2007).

Segundo Valente(2007), um caso de uso essencial é abstrato e não faz menção à tecnologia a ser utilizada. Note que o exemplo a seguir de caso de uso essencial e numerado é completamente desprovido de características tecnológicas. Esse caso de uso vale para a situação em que o sistema tem mais de uma interface para a mesma funcionalidade. Por exemplo, uma interface no site na Internet e outra interface via telefone celular:

1. Cliente fornece sua identificação.
2. Sistema identifica o usuário.
3. Sistema fornece operações disponíveis.
4. Cliente solicita o saque de uma determinada quantia.
5. Sistema fornece a quantia desejada da conta do Cliente.
6. Cliente recebe dinheiro e recibo.

Diferentemente, em um caso de uso real, as descrições das interações citam detalhes da tecnologia a ser utilizada na implementação. A descrição em um grau de abstração real se compromete com a solução de projeto (tecnologia) a ser utilizada para implementar o caso de uso.
Uma boa prática é utilizar a "**regra prática dos 100 anos**" para verificar se o caso de uso contém algum detalhe de tecnologia. Questione se a narrativa seria válida tanto 100 anos atrás, como daqui a 100 anos. Se a resposta for positiva, então muito

provavelmente esse caso de uso é essencial. Caso contrário, trata-se de um caso de uso real (VALENTE, 2007).

Formas de representação de caso de uso

Diagrama de Casos de Uso

A maior dificuldade em modelarmos um sistema, está nos requisitos que devemos gerenciar, não está nos diagramas que temos que desenhar, no código que devemos criar ou nas bases de dados que devemos projetar.

O modelo de casos de uso é uma representação gráfica dos procedimentos externamente observáveis do sistema e dos elementos externos que o sistema interagem com ele.

Um Modelo de Casos de Uso molda os requisitos funcionais do sistema. O diagrama da UML utilizado na modelagem de casos de uso é o diagrama de casos de uso.

A técnica de modelagem através de casos de uso foi criada por um engenheiro de software que projetava um sistema telefônico, na Suécia nos anos 70.

Depois, Jacobson incorporou esta técnica para um processo de desenvolvimento de software denominado Objetory. Posteriormente, ele se uniu a Booch e a Rumbaugh, e a descrição de casos de uso foi incorporada à Linguagem de Modelagem Unificada (VALENTE, 2007).

Este modelo popularizou-se na documentação de requisitos funcionais de uma aplicação devido a sua descrição gráfica e linguagens diretas, simplificando assim a interação de desenvolvedores e usuários.

Ainda segundo Valente (2007), o modelo de caso de uso é um dos mais importantes da UML. Ele direciona diversas tarefas posteriores ao ciclo de vida do sistema de software. Além disso, o modelo de casos de uso força os desenvolvedores a moldarem o sistema conforme a visão dos usuário, e não o usuário de acordo com o sistema. O modelo de casos de uso é composto de: casos de uso, atores e relacionamento entre estes. Vejamos maiores detalhes de cada um a seguir.

Caso de Uso – Atores e Relacionamentos

Na terminologia da UML, qualquer elemento externo que interage com o sistema é denominado ator. Vamos detalhar melhor essa definição. O termo “externo” indica que atores não fazem parte do sistema. E o termo “interage” significa que um ator troca (envia e/ou recebe) informações com o sistema.

6 DEFINIÇÃO DE CASO DE TESTE DE SOFTWARE

Em geral, define-se formalmente um caso de teste como a especificação mais detalhada do teste, com a pormenorização de campos de telas, formulários, etc. Estabelecem-se quais informações serão empregadas durante os testes dos cenários e quais serão os resultados esperados. Para isso, é necessário determinar a massa a ser utilizada no teste de modo a validar todos os requisitos do software.

Segundo Coutinho (2011) um bom caso de teste deve conter:

- identificação das condições de testes;
- precondições;
- poscondições;
- critério de aceitação;
- identificação dos casos de testes (o que testar);
- detalhamento da massa de entrada e de saída;
- critérios especiais, casos necessários, para a geração da massa de teste;
- especificação das configurações de ambiente no qual o teste será executado: sistema operacional, ferramentas necessárias, origem dos dados etc.(onde testar);
- definir o tipo de implementação do teste: automática/manual (como testar).
- definir o cronograma, ou seja, em qual fase esse teste será executado (quando testar);
- listar as interdependências, caso existam, entre os casos de teste.
-

Os fatores motivadores para escrever um bom caso de teste são:

- o requisito a ser verificado como o input do teste;
- a configuração do ambiente no qual o teste deve ser executado;
- o tipo de implementação (manual ou automática);
- o momento do teste, ou melhor, a fase do ciclo de desenvolvimento na qual um determinado teste deve ser executado.

O caso de teste deve ter as características a seguir para que possa ser usado e atender às expectativas de validação da qualidade:

- efetivo: testar o que se planejou testar;

- econômico: sem passos desnecessários;
- reutilizável: que possa ser repetido;
- rastreável: que possa identificar o requisito a ser testado;
- autoexplicativo: que possa ser testado por qualquer testador.

6.1 DERIVAÇÃO DE CASO DE TESTE

Os casos de teste para a técnica do teste funcional costumam derivar de uma especificação formal (caso de uso, etc.). É necessário desenvolver casos de teste para cada cenário de caso de uso. Os cenários de caso de uso são identificados a partir da descrição dos caminhos que percorrem os fluxos básicos e alternativos.

No diagrama de caso de uso do RUP (Figura 12), por exemplo, os vários caminhos que atravessam o caso de uso refletem o fluxo básico e os fluxos alternativos que são representados pelas setas.

O fluxo básico ou principal, representado pela linha preta e retilínea, é o caminho mais simples através do caso de uso. Cada fluxo alternativo começa no fluxo principal e depois, de acordo com uma condição específica, é executado. Os fluxos alternativos podem retornar ao fluxo básico (por exemplo, os fluxos alternativos 1 e 3 da Figura 12), podem se originar de outro fluxo alternativo (fluxo alternativo 2) ou terminar o caso de uso sem retornar a um fluxo (fluxos alternativos 2 e 4).

Após percorrer cada caminho possível através do caso de uso mostrado no diagrama, é possível identificar os diversos cenários de caso de uso. Dessa forma, começando pelo fluxo básico, este é combinado com os fluxos alternativos, com as exceções e as regras de negócios descritas no caso de uso.

Com isso, podem-se elaborar os cenários de teste com base nos cenários do caso de uso.

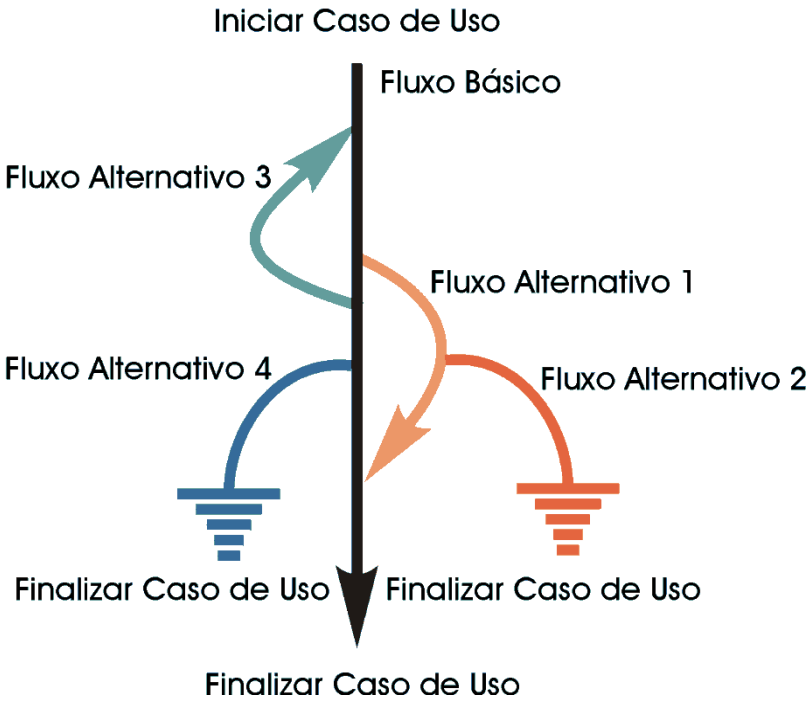

Figura 12 –. diagrama de caso de uso do RUP.
Fonte: Bastos (2012).

No entanto, nem todos os requisitos que darão origem aos cenários de teste são extraídos dos casos de uso. Os requisitos não funcionais - como requisitos de desempenho, segurança e acesso - e os requisitos de configuração que especificam comportamentos ou características adicionais do teste são obtidos a partir da Especificação Suplementar.

Para os testes de interface, usabilidade, etc., a fonte de consulta é a documentação de Padrões de Interface ou especificações de usabilidade.

Sendo assim, concluímos que se existir uma especificação de referência, e, caso não exista, a informação deve ser conseguida com o próprio cliente, com o objetivo de fazer do teste uma validação do especificado versus o que foi implementado.

6.2 UTILIZANDO CASOS DE USO NA ELABORAÇÃO DE CASO DE TESTE

Os diagramas de casos de uso, são construídos em forma de cenários, contendo caminho principal, alternativo e de exceção, pré-condição, pós-condição e passos a serem seguidos pelo sistema. Na elaboração do fluxo de análise de requisitos de um projeto de software, ele descreve a visão do usuário em relação às funcionalidades do sistema. O caso de uso tem um papel muito importante na atividade de teste, pois

se o projeto de teste for baseado em cenários, ele vai se concentrar no que o usuário faz e não no que o produto faz. Isso significa que as tarefas podem ser detectadas através dos casos de uso existentes. A partir do caso de uso, a equipe de teste pode criar uma visão estratégica e os testes necessários para trabalhar cada cenário descrito no caso de uso (TARGET TRUST, 2005).

A Figura 13, O caso de uso pode ter um papel muito importante na atividade de teste, pois se o projeto de teste for baseado em cenários, ele vai se concentrar no que o usuário faz e não no que o produto faz. Primeiramente, são analisados os casos de uso existentes; após esta análise são criados os possíveis cenários de teste, a partir destes são elaborados os casos de teste.

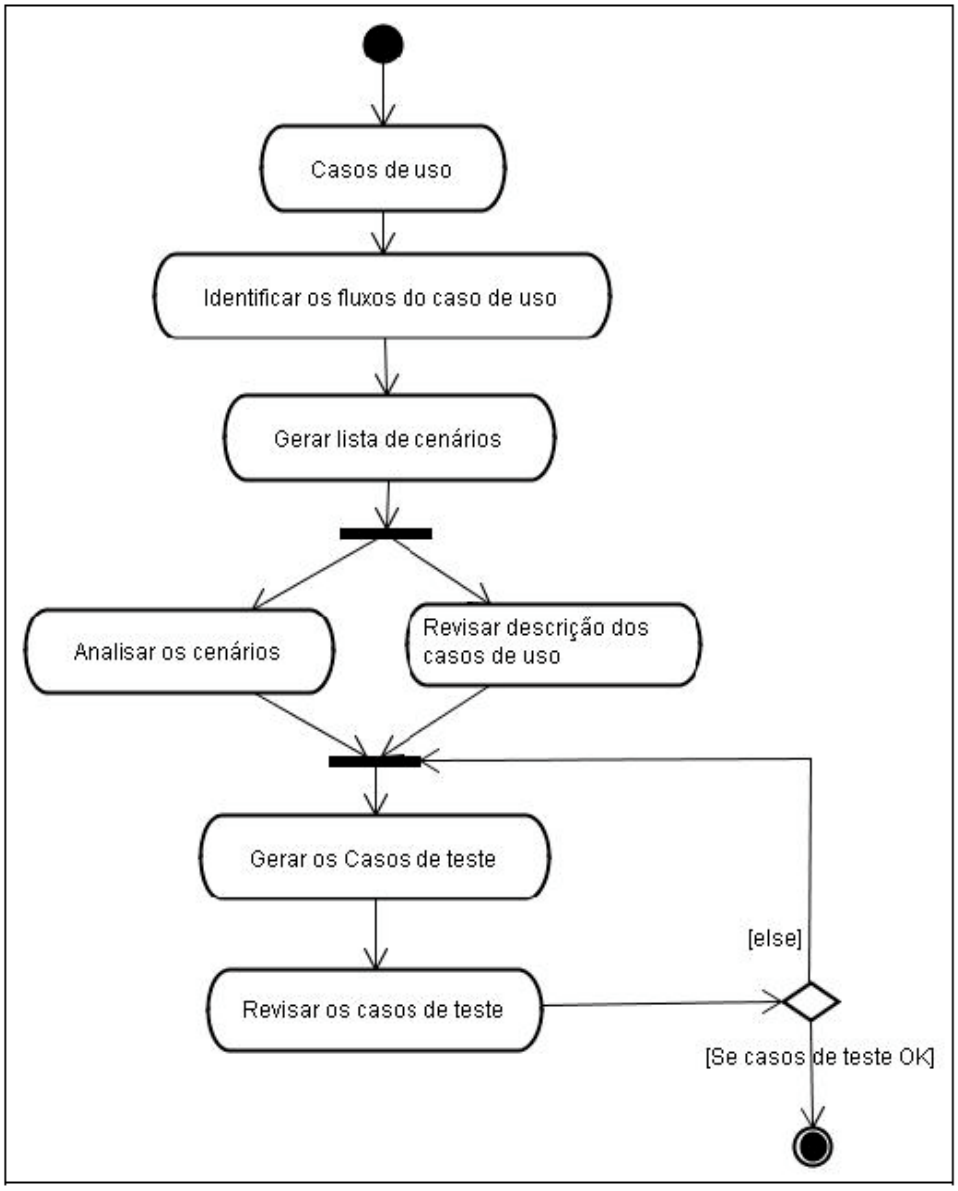

Figura 13 – Diagrama de atividades.
Fonte: Targetust (2005).

Segundo Heumann (2001), a geração de casos de teste a partir de casos de uso é eficiente e ajuda na cobertura completa dos testes. Sua metodologia analisa as partes consideradas mais importantes na descrição de um caso de uso para a geração de casos de teste que são: o fluxo principal e o alternativo dos eventos. O autor afirma que a geração pode ser feita a partir da execução de três passos:

1. Para cada caso de uso, gerar uma lista completa de cenários de casos de uso.
2. Para cada cenário, identificar pelo menos um caso de teste e as condições para sua execução.
3. Para cada caso de teste, identificar os dados que de uso, identificar o fluxo principal e os fluxos alternativos e criar uma tabela com os possíveis cenários.

No segundo passo, identificar os possíveis casos de teste para cada cenário. Para isso, devem-se analisar os cenários e as descrições de cada caso de uso. Após gerar uma outra tabela com a identificação do caso de teste, descrevê-lo incluindo o cenário a ser testado, dados de entrada para a implementação do teste e, por final, a saída esperada.

No terceiro passo, deve-se revisar e validar os casos de teste para assegurar sua acura cidade e para identificar casos de teste redundantes ou faltantes. Depois se aprovados, os dados de teste devem ser identificados, pois casos de teste sem dados, não podem ser executados.

Segundo Bastos (2012), as derivações de casos de teste funcional costumam derivar de uma especificação formal (caso de uso, etc.). É necessário desenvolver pelo menos um caso de teste para cada cenário de caso de uso. Sendo que os cenários podem ser identificados a partir da descrição do fluxo básico e alternativo de cada caso de uso. Após percorrer cada caminho é possível identificar os diversos cenários de caso de uso. Dessa forma, começando pelo fluxo básico, após os alternativos, com as exceções e as regra de negócio descritas no caso de uso é possível elaborar os cenários de teste com base nos cenários do caso de uso.

No entanto, nem todos os requisitos que darão origem aos cenários de teste são extraídos do caso de uso. Os requisitos não funcionais como, por exemplo, requisitos de desempenho, segurança e outros, podem ser extraídos a partir de especificação suplementar.

7 CONCLUSÃO

Este trabalho envolveu o estudo de Qualidade de Software, Conceito de testes de software, Planejamento em teste de Software, Caso de Uso, Definição de Caso de teste de Software e teve como principal objetivo propor como desenvolver um software e obter maior qualidade utilizando um modelo de apoio no planejamento de testes de software com ênfase em testes funcionais possibilitando assim a geração de casos de testes a partir de casos de uso, pois a geração de caso de uso se faz no levantamento dos requisitos e funcionalidades no início dos trabalhos de modelagem de um sistema. A necessidade de investigações nestes conceitos e ambientes possibilitou uma grande coleta de informações.

Pudemos destacar que a busca constante na qualidade de software não se faz apenas no começo do desenvolvimento de software ou no seu final realizando testes como uma atividade, mas sim como um processo que visa toda a engenharia de software (BASTOS, 2012). Concluímos que só assim se poderá estar dentro das expectativas de qualidade de software, e também verificamos que é fundamental um bom gerenciamento de qualidade para que os custos e prazos estejam dentro das expectativas dos clientes, conscientizando-os que mesmo tendo um custo e prazo menor oferecidos por outras empresas no mercado de desenvolvimento de Software é de suma importância um planejamento, acompanhamento e controle de qualidade durante todo o período de desenvolvimento do sistema para que as divergências ou defeitos sejam descobertos e resolvidos bem antes de colocar o sistema em produção evitando enormes problemas e de difícil solução gerando grandes prejuízos no mercado de atuação das empresas (BASTOS, 2012).

Mostramos nos conceitos de Testes as definições de Testes de Verificação que envolve os testes unitários, integração e de sistema, nos quais se verifica se o software está sendo construído corretamente (COUTINHO, 2011). Os Testes de Validação permitem validar se o software está fazendo o que foi definido nos requisitos. Os Testes Caixa Branca ou estrutural são a garantia que o sistema esteja estruturalmente robusto. Os Testes Caixa Preta ou Funcional do sistema são desenhados para garantir que os requisitos e as especificações do sistema tenham sido atendidos. O Nível de teste mostra em que fase do desenvolvimento se aplica

um determinado teste que pode ser: teste de unidade, teste de integração, teste de sistema e teste de aceitação (BASTOS, 2012).

Apesar do grande avanço tecnológico e melhoria das ferramentas em programação e modelagem de sistemas, ainda é grande o desconhecimento dos profissionais da área de TI nos testes de software em seus conceitos. Os mesmos estão acostumados a atuarem em empresas de fábrica de softwares nas quais os testes de software ainda são aplicados apenas como uma fase ou tarefa inicial ou final, um pouco antes ou durante a implantação do sistema em produção, e não como um processo que envolve um gerenciamento com planejamento, acompanhamento e controle para investigação e detecção de defeitos ou divergências desde os levantamentos dos requisitos e funcionalidades do sistema no início até o final dos trabalhos (BASTOS, 2012).

Vimos que o planejamento em teste de software se faz a partir da geração um plano de teste de software, que é um documento padronizado pela empresa com normas internacionais descritas no IEEE 829 e QAI, com estrutura que define a abordagem e nível de cobertura dos testes e tem como base os requisitos do sistema e requisitos dos testes (BASTOS, 2012).

Verificamos em nossa pesquisa que a melhor solução no planejamento do projeto de teste e na geração de um plano de teste que definirá a abrangência e nível de cobertura de testes será utilizar o modelo dos Planos específicos do Projeto do PMBOK, observando as normas do IEEE 829, QAI e elementos mais críticos caracterizados pelos usuários do software (BASTOS, 2012).

Neste trabalho mostramos também que o caso de uso é uma especificação de interações entre o sistema e os agentes externos que utilizam o sistema e seu diagrama, muito utilizado na análise de requisitos de um projeto de software descrevendo a visão do usuário em relação às funcionalidades do sistema em forma de cenários. O analista de teste pode gerar os casos de testes a partir dos cenários de testes que são baseados nos cenários dos casos de uso (VALENTE, 2007).

Assim com este trabalho concluímos com segurança que os casos de testes consistentes são gerados a partir dos casos de uso utilizando seus requisitos e cenários funcionais e têm um papel importante na conscientização de elaboração de modelo de apoio ao planejamento de testes de software com qualidade.

Para trabalhos futuros, seria interessante uma apresentação e utilização de ferramentas disponíveis no mercado para organizar e gerenciar o processo de teste,

pois elas possuem as etapas para o planejamento e execução dos testes e acompanhamento das execuções. Além disso, poderão contribuir para a reexecução, reaproveitamento de planos e casos existentes, na execução de futuros testes. Facilitariam também o dia a dia do analista de teste, pois elas contemplam a possibilidade de gerar casos de teste de forma semiautomática.

8 REFERÊNCIAS

BASTOS, A. et. al . **Base de Conhecimento em Teste de Software**. 3 ed. São Paulo: Martins, 2012.

COUTINHO, Pedro Henrique Mannato. **Teste de Software**, Apostila ESAB (Escola Superior Aberta do Brasil). Vila Velha, 2011.

HEUMANN, Jim. EMBRAPA - **Teste baseado em caso de uso**. Disponível em: <http://www.infoteca.cnptia.embrapa.br/bitstream/doc/8852/1/bp10.pdf>. Acesso em: 10 de agosto 2013.

IEEE Standard Glosary of Software Engineering Terminology" IEEE, Piscatway, NJ std 610.12-1990, 1990.

ISTQB International Software Testing Qualification Board. **Glossário Padrão de Termos Utilizados em Testes de Software.** Versão 2.1br.Scrib: Disponível em: <http://pt.scribd.com/doc/56113819/ISTQB-Glossario-V-2-1br>. Acesso em agosto de 2013.

JORGENSEN, Paul, TEUNISSEN, Ruud. , VEENENDAAL, Erik van. **Software testing: A guide to the TMap approach**,. Boston: Addison-Wesley, 2002.

SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Addison Wesley, 2010. 8ª edição

TARGET TRUST. **Introdução ao processo de teste de software**. Porto Alegre: 2005. 80 p.

UML: UML: Disponível em http://www.uml.com acesso em 10 de agosto de 2013.

VALENTE, Carlos. **Metodologia de Análise de Sistemas**, Apostila ESAB (Escola Superior Aberta do Brasil). Vila Velha, 2007.