FACULDADE DE TECNOLOGIA DE SÃO BERNARDO DO CAMPO
“ADIB MOISÉS DIB”

GISELE DA MATTA GONÇALVES
GUILHERME VEIGA GASPAR DA SILVA
ROBERTO SARTORI MARTINS
VALDEMIR JOSÉ PIRES

GUIA RÁPIDO DE TESTES DE SOFTWARE

São Bernardo do Campo – SP
Novembro/2015

GISELE DA MATTA GONÇALVES
GUILHERME VEIGA GASPAR DA SILVA
ROBERTO SARTORI MARTINS
VALDEMIR JOSÉ PIRES

GUIA RÁPIDO DE TESTES DE SOFTWARE

Trabalho de conclusão de curso apresentado a Faculdade de Tecnologia de São Bernardo do Campo Adib Moisés Dib como requisito parcial à obtenção do título de tecnólogo em Informática para Negócios.

Orientadora do projeto:
Profa. Me.Simone Faccio

São Bernardo do Campo – SP
Novembro/2015

GISELE DA MATTA GONÇALVES
GUILHERME VEIGA GASPAR DA SILVA
ROBERTO SARTORI MARTINS
VALDEMIR JOSÉ PIRES

GUIA RÁPIDO DE TESTES DE SOFTWARE

Trabalho de conclusão de curso apresentado a Faculdade de Tecnologia de São Bernardo do Campo “Adib Moisés Dib” como requisito parcial para a obtenção do título de tecnólogo em Informática para Negócios.

Trabalho de Conclusão de Curso apresentado e aprovado em: 23/11/2015.

Banca examinadora:

_____
Profa. Me Simone Faccio, FATEC SBC – Orientadora

_____
Prof. Esp. Edmilson S. Carvalho, FATEC SBC - Avaliador

_____
Profa. Me. Rosangela Kronig, FATEC SBC – Avaliadora

Este trabalho é dedicado a Deus, as nossas famílias e aos nossos professores que nos orientaram e apoiaram.

Agradecemos aos Professores da FATEC SBC pela dedicação e ensinamentos transmitidos que permitiram que o trabalho fosse realizado, principalmente à nossa Orientadora, a Professora Mestra Simone Faccio e ao Professor Edmilson S. Carvalho; aos nossos amigos que contribuíram com materiais de pesquisa, que possibilitaram que nós conseguíssemos finalizar este trabalho.

“Testes podem apenas mostrar a presença de erros e não a sua ausência”

EDSGER WYBE DIJKSTRA (s.d.)

RESUMO

Este trabalho propõe a criação de um guia rápido de teste de software, destinado aos profissionais, professores, estudantes e demais pessoas interessadas em aprender mais sobre essa área do conhecimento. O trabalho pretende apresentar e orientar o melhor caminho a ser seguido na elaboração e execução dos testes, visando eficiência, exatidão, segurança, rapidez e redução de custos adicionais derivados de defeitos e falhas. O trabalho pretende ainda estabelecer a ligação dos testes de softwares à qualidade do produto final, abrangendo desde o início dos testes, as técnicas utilizadas em sua execução e os benefícios para as empresas que adotam os testes no seu cotidiano. Após verificar que em algumas empresas, principalmente as micros e pequenas, que há uma lacuna referente à quais testes devem ser aplicados, foram identificados durante a pesquisa que existe uma grande vantagem em utilizar um guia de rápida visualização das tarefas, mostrando um caminho a ser seguido na execução de teste, cuja importância é fundamental para a qualidade do software, diminuindo os erros e defeitos que contribuam para o aumento de sua credibilidade. Para o desenvolvimento deste guia, foram pesquisadas várias ferramentas e verificou-se que a melhor que se adequava ao projeto foi a ferramenta TestLink, que além de ser de fácil utilização, é gratuita (open source) e ainda com larga adequação ao projeto de um software construído por micro e pequenas empresas.

Palavras-chave: Software. Guia. Defeitos. Testes. Qualidade.

ABSTRACT

This paper proposes the creation of a short guide to software testing, for practitioners, teachers, students and other people interested in learning more about this area of knowledge. The paper intends to present and guide through the best way forward in the development and execution of the tests, aiming at efficiency, accuracy, security, speed and reduction of additional costs motivated by defects and failures. The paper also points match software tests and the quality of the final product, ranging from the early tests, the techniques used in its execution and benefits for companies that adopt the tests in their daily lives. After verifying that in some companies, especially the micro and small, was found there is a gap related to what tests should be applied, they have been identified during the survey that there is a great advantage to use a Quick View tab of tasks, showing a path to followed in the test run, whose importance is fundamental to software quality, reducing errors and failures that contribute to the increase of its credibility. To develop this guide, we were surveyed several tools and found out that the best choice for the project was TestLink tool, which besides being easy to use, is free (open source) and with wide adaptation to the design of a software built by micro and small businesses.

Keywords: Software. Guide. Defects. Tests. Quality.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

BIOS	Basic Input Output System (Sistema Básico de Entrada/Saída)
CD	Compact Disk (Disco Compacto)
CPU	Central Processing Unit (Unidade Central de Processamento).
CRT	Cathodic Ray Tube (Tubo de raios catódicos)
DRAM	Dinamic Random Acess Memory (Memória Dinamica de Acesso Aleatório)
DVD	Digital Versatile Disc (Disco Digital Versátil)
HD	Hard Disk (Disco Rígido)
IDE	Integrated Driver Eletronics (Controlador Eletrônico Integrado)
IEC	International Electrotechnical Commission (Comissão Eletrotécnica Internacional).
ISO	International Organization for Standardization (Organização Internacional para Padronização).
LCD	Liquid Crystal Display (display de cristal líquido)
LED	Light Emitting Diode (Diodo Emissor de luz)
NATO	NorthAtlantic Treaty Organization (Organização do Tratado do Atlântico Norte).
RAM	Random Access Memory (Memória de acesso aleatório)
ROM	Read Only Memory (Memória Somente de Leitura)
RPM	Revolutions per Minute (Rotações por Minuto)
SaaS	Software as a Service (Software como Serviço).

SGBDs	Sistema Gerenciador de Banco de Dados.
SRAM	Static Random Acess Memory (Memória Estática de Acesso Aleatório)
SquaRE	Product Quality Requirements and Evaluation (Requisitos de Qualidade Avaliação de Produtos de Software).
XML	eXtensible Markup Language, (Linguagem de Marcação Extensível Recomendada pela W3C).
TI	Tecnologia da Informação.
USB	Universal Serial Bus (Barramento Serial Universal)

SUMÁRIO

INTRODUÇÃO

A qualidade no desenvolvimento de um software tem como premissa básica os testes, que devem ser realizados com vistas a verificar os diversos itens que podem resultar em um diferencial quando da sua efetiva utilização. Nesse sentido, Bastos et al. (2012) explanam sobre a existência de três fatores preponderantes da qualidade de software: Confiança, Funcionalidade e Performance. Esclarecendo ainda, que durante as décadas de 1970, 1980 e 1990, os testes de software eram efetuados pelos próprios desenvolvedores e que as informações extraídas desses testes eram muito simples, não permitindo que todos os defeitos fossem detectados.

Muitas vezes os próprios usuários eram responsáveis por efetuar os testes quando o sistema já estava em fase produção, muitos defeitos eram encontrados em um momento que o custo com a correção desses defeitos torna-se muito oneroso para o desenvolvedor.

Evidencia-se que, o mercado global e a forte concorrência, que se verifica na atualidade, provoca uma reflexão que culmina com a constatação de que muitas vezes apenas ter uma boa estratégia, não é o suficiente para direcionar as organizações para seus objetivos. São necessários fortes investimentos em tecnologias e recursos que promovam valor para seus produtos e serviços oferecidos. Quando a empresa consegue agregar valor aos produtos e serviços, ela é capaz de adquirir vantagem competitiva frente aos concorrentes. Essa vantagem fará toda a diferença ao lidar com os contratempos e imprevistos do mercado.

Segundo Bastos et al. (2012), com a crescente complexidade dos sistemas e o surgimento da Internet, a imagem das empresas passou a ser cada vez mais exposta ao grande público. Isso levou as organizações a procurar novos caminhos para melhorar a qualidade dos seus softwares. Um desses caminhos foi o aprimoramento da atividade de testes trazendo resultados imediatos para a qualidade do produto a ser entregue. Foi desta forma que cada vez mais empresas passaram a adotar um modelo de testes, significando em um curto prazo, a redução nos custos de manutenção do software.

Apesar de existirem várias técnicas, políticas e metodologias na área de teste de software, no âmbito das micro e pequenas empresas existe uma deficiência perceptível em formas específicas para tais, em especial, no segmento de processos de teste de software e ferramentas que o apoiem de forma integrada (FAPEG, 2013).

Seguindo esta premissa, pretende-se neste projeto estabelecer a ligação dos testes de softwares à qualidade do produto final, abrangendo desde o inicio dos testes, as técnicas utilizadas em sua execução e os benefícios para as empresas que adotam os testes no seu cotidiano, e ainda, pretende-se de forma simples, mas ricamente detalhada e de fácil acesso, criar um guia de referência rápido para a elaboração e execução de testes de softwares, destinado tanto aos profissionais e professores, como aos estudantes e demais pessoas interessadas em aprender mais sobre essa área do conhecimento, o caminho que pode ser seguido na elaboração e execução dos testes, possibilitando que o trabalho possa ser feito com exatidão e em menor tempo, visando assim, economia ao invés de custos adicionais.

O público alvo desse guia é o profissional, cujo escopo de atividades está relacionado ao desenvolvimento e engenharia de software, tais como analistas desenvolvedores e responsáveis pelos testes e qualidade do software em pequenas empresas. Esse guia pretende estabelecer padrões e diretrizes primordiais para teste de software de modo a facilitar, melhorar a qualidade e reduzir os custos e tempos alocados as atividades de teste.

De acordo com as normas da Instituição, este presente projeto será desenvolvido em duas etapas. A primeira etapa foi desenvolvida no decorrer do 5º ciclo do curso de Informática para negócios e tem como premissa básica a fundamentação teórica, a qual será baseada em pesquisas bibliográficas, artigos científicos e monografias. Após a leitura e analise do conteúdo serão extraídas informações pertinentes para desenvolvimento do projeto.

Sendo assim, o primeiro capítulo, fundamentação teórica, foi dedicado para a definição de conceitos importantes no ambiente de teste e qualidade de software. Além de abordar as características dos diferentes tipos de testes, suas perspectivas e suas respectivas áreas de atuação.

O segundo capítulo, Metodologia, também foi executado durante este ciclo e, nessa etapa foram definidos os materiais utilizados e quais os métodos que serão aplicados durante o desenvolvimento do projeto que ocorrerá durante o 6º ciclo do curso, o qual estará vinculado ao cumprimento do cronograma estipulado que será apresentado no fim deste capítulo.

O terceiro e último capítulo, que será executado no decorrer do 6º ciclo do curso, é direcionado para a elaboração de um relatório técnico, que tem como premissa subsidiar o acompanhamento das etapas de desenvolvimento do projeto, com vistas a sustentar de maneira adequada a aplicação das teorias, ideias e conceitos levantados anteriormente e que devem resultar na conclusão do projeto conforme proposto.

Para finalizar o projeto serão apresentadas as considerações finais, com o objetivo de evidenciar os ganhos alcançados com as pesquisas realizadas e redigir comentários sobre a etapa do desenvolvimento que reuniu desde a teoria que foi utilizada até os aspectos práticos, os quais podem vir a contribuir de maneira importante na qualificação que se busca no curso de Informática para Negócios.

1 FUNDAMENTAÇÃO TEÓRICA

Com a popularização da Internet, o grande público teve cada vez mais acesso direto às empresas, podendo compara-las entre si, isto levou estas organizações a procurarem novas formas para melhorar a qualidade de seus projetos, e uma destas formas foi o aperfeiçoamento na execução dos testes de software, e neste capítulo são abordados os conceitos essenciais ao entendimento dos objetivos deste projeto.

1.1 Tecnologia

Para Sancho (1998 apud BRIGNOL, 2004) até o século XX a tecnologia era pensada desta forma: a tecnologia é como um corpo de conhecimentos que além de usar método científico, cria ou transforma processos materiais.

1.1.1 Conceitos e definições

A palavra tecnologia tem origem no grego ‘tekhne’ que significa ‘técnica, arte ou ‘ofício’ juntamente com o sufixo ‘logia’ que significa ‘estudo’. Reforçando este conceito, o dicionário Houaiss (2010), complementa que: Tecnologia é um conjunto de conhecimentos científicos, dos processos e métodos usados na criação e utilização de bens e serviços. Técnicas ou conjunto de técnicas de um domínio particular. Tecnologia de ponta - aquela que se utiliza de técnicas de última geração.

Tecnologia, no sistema econômico capitalista, induz ao pensamento de que é ‘algo físico’, ou seja, a estrutura econômica é que nos leva a crer que ‘tecnologia é igual a objeto’, onde objetos podem ser comprados, adquiridos e deixam de ser usados muito rapidamente, gerando assim um imenso lucro. Desta forma, torna-se muito importante para a continuidade deste sistema econômico que se continue a pensar que tecnologia se resume a objetos e, portanto, se pode comprar, pode ter

tecnologia (BRITO; KNOLL, 2013, grifos do autor). Segundo as autoras, este pensamento não é o correto, pois tecnologia não é tão somente um objeto. Ela pode sim ser identificada por ou através de um objeto, desde que se tenha clareza de que tecnologia não é só isso ou apenas algo físico. Tecnologia não se resume apenas a computadores, aparelhos celulares, internet, cabos, redes, tablets, carros, aviões, canetas… ou qualquer outro objeto que possa ser adquirido. Tecnologia é muito mais do que qualquer produto que se compra, pois, tecnologia é conhecimento. O conceito de tecnologia, na sociedade, é fruto da história, do contexto cultural, social e econômico e, principalmente, da formação profissional.

Outra definição interessante reforça que, atualmente a tecnologia é sinônima de aparelhos cada vez mais inteligentes, sofisticados e rápidos, como o computador, tablet ou smartphone. No entanto, é correto afirmar que um arco e flecha, por exemplo, também é considerado tecnologia. O autor questiona, afinal de contas, quem estará correto nesta história? O conceito abordado pelo dicionário vai direto ao assunto e traz um resumo que não deixa muito claro o que é a tecnologia. Assim, pode se dizer que a tecnologia é o uso de técnicas e do conhecimento adquirido para aperfeiçoar ou facilitar o trabalho com a arte, a resolução de um problema ou a execução de uma tarefa específica. Dessa forma, ela pode ser aplicada em diversas tarefas diferentes, aparecendo em situações que poucas pessoas consideram envolver a tecnologia. O simples aproveitamento dos recursos naturais e a transformação do ambiente ao seu favor, por exemplo, é capaz de ser considerado como um movimento tecnológico (KARASINSKI, 2013).

Segundo Karasinski (2013) o fato é que a tecnologia sempre existiu ao longo da evolução humana inclusive confundindo-se com a própria história, a evolução da tecnologia se confunde com o progresso do próprio homem, desde a descoberta do fogo até os dias atuais.

Albagli e Lastres (1999) afirmam que as tecnologias afetam todas as atividades econômicas: setores maduros, como o têxtil, se rejuvenescem; surgem novas organizações. No centro dessas mudanças encontra-se o crescimento cada vez mais acelerado dos setores intensivos em informação e conhecimento.

1.1.2 Tecnologia da Informação

A Tecnologia da Informação (TI) é um tema muito amplo e com várias aplicações. Antes de tudo, informação é, de acordo com Alecrim (2013), um patrimônio, é algo que possui valor. Quando digital, não se trata apenas de um ‘amontoado’ de bytes aglomerados, mas sim de um conjunto de dados classificados e organizados de forma que uma pessoa, uma empresa ou qualquer outra entidade possa utilizar de acordo com sua necessidade ou objetivo.

A TI oferece as ferramentas para permitir que toda a empresa solucione problemas cada vez mais complexos e também para que aproveitem as oportunidades que contribuem para o sucesso da empresa (POTTER; RAINER; TURBAN, 2005).

A TI é um setor estratégico das organizações atuais, necessitando de qualidade na gestão dos serviços, engajando-se também na busca por normatização dos processos, sendo assim todas as necessidades atuais e futuras dos usuários deve ter como objetivo a qualidade, e ainda que deva se incorporar a moderna TI, à dinâmica das organizações para conseguir o sucesso organizacional, segundo Chiavenato (2003).

De acordo com Foina (2001), foi com o advento dos computadores nas empresas e organizações que a TI surgiu. Antes, os processos de tratamento das informações eram em formatos de memorandos, planilhas e tabulações, todas datilografadas e distribuídas por meio de malotes aos funcionários. Analisando os avanços da TI percebe-se o quanto esse instrumento de tomada de decisão é importante no mundo dos negócios, nas empresas e na própria tecnologia.

Segundo Keen (1996, p. 37), “a maior evolução técnica dessa época foi a passagem do processamento de transações para o gerenciamento de banco de dados". Surgiram então os sistemas gerenciadores de banco de dados (SGBDs), que organizam as informações de uma maneira eficaz, evitando duplicidade e facilitando sua análise.

“A TI é reconhecida como fator crítico de capacitação, principalmente através das telecomunicações, que permite eliminar barreiras impostas por local e tempo às atividades de coordenação, serviço e colaboração” (KEEN, 1996, p. 49). Concluindo esta linha de pensamento, de uma maneira muito rápida, a mudança se acelerou em quase todas as áreas dos negócios e das tecnologias. A transformação e utilização das ferramentas da TI se tornam globais e as distinções entre computador e comunicação desaparecem mudando radicalmente o mundo dos negócios, fazendo com que as distâncias desaparecessem. O computador se torna elemento de TI indispensável em uma empresa.

Toda tecnologia representa recursos que podem ser compartilhados por toda a empresa e este compartilhamento faz parte da infraestrutura de TI. A infraestrutura de TI provê a base onde a empresa monta o seu sistema de informação específico que, por exemplo, pode aumentar a flexibilidade das organizações, ao ponto que grandes organizações poderem usar a TI para adquirir um pouco da agilidade e da rapidez de resposta das organizações de pequeno porte e um aspecto deste fenômeno é a personalização em massa, que é a capacidade de oferecer produtos ou serviços personalizados individualmente em grande escala. Esta tecnologia pode tornar os processos de produção mais flexível, permitindo que os produtos sejam personalizados segundo as exigências de cada cliente (ZIPKIN, 2001 apud LAUDON; LAUDON, 2004).

Como exemplo a indústria automobilística, que nos primórdios da produção em série, o cliente podia escolher qualquer cor, desde que fosse o preto, e hoje a personalização individual atinge vários itens do automóvel: “Com o lançamento do Novo Uno, a Fiat também introduziu o conceito de carro 100% personalizável” (www.icarros, 2012).

Laudon e Laudon (2004) esclarecem que as empresas cada vez mais podem usar as ferramentas de TI para criar uma infraestrutura capaz de coordenar suas atividades ou setores. Habilitando as organizações a reduzir radicalmente seus custos de agências e transações, esta nova infraestrutura proporciona ampla plataforma para comércio e negócios eletrônicos e para a empresa digital

emergente. Ela é baseada em redes poderosas com tecnologia de Internet, XML[1] e Java[2] que provêm conectividade e integração de aplicativos limitados, pois muitas empresas têm aplicativos importantes, cujo hardware, software e componentes de redes diferentes precisam ser integrados por outros meios. Este é um dos motivos para execução de testes, garantindo que esta integração seja feita com a qualidade que o cliente espera. Assim a Internet, que é a mais bem conhecida e a maior implementação de trabalho em rede, dispõe de uma gama de recursos em TI, utilizada amplamente por várias organizações. Os autores concluem afirmando que implementar redes empresariais integradas e a nova infraestrutura de TI, tem criado problemas, e também oportunidades para as organizações que conseguirem desenvolver produtos com garantia de qualidade.

Segundo Mañas (2011) estes são alguns dos desafios que os técnicos terão que superar com a construção eficaz de softwares que consigam indexar as redes sem conflito, para que as organizações possam cada vez mais ampliar seus ganhos. Os cientistas, os práticos e inclusive até os leigos conseguem distinguir entre o que são informação e o que são instruções, nos documentos que fazem parte de instrumentos ou equipamentos que auxiliam a produção, sobretudo no setor de informática. Ainda de acordo com o autor, paralelamente aparecem em algumas organizações, novos postos de responsabilidade, que são direcionados pela área de informática, ligada, sobretudo, ao da informação, pois são os que organizam, depois os que lidam com os sistemas de informação e mais recentemente, os responsáveis pela gestão da informação. Esta nova política de informática, deve estar associada a uma política de informação, sendo a primeira responsabilizada aos especialistas em informática, que cuidariam da estrutura física da manutenção e necessidades de equipamentos, e a outra a um novo gênero de especialista, o que daria suporte, com a concepção e geração de conteúdos informacionais.

---

[1] **XML:** (EXTENSIBLE MARKUP LANGUAGE - Linguagem de marcação extensível) recomendada pela W3C (LAUDON; LAUDON, 2004).

[2] **JAVA:** Linguagem de programação orientada a objetos desenvolvida pela Sun Microsystems na década de 1990 (www.java.com, 2015)

1.2 Hardware

O Hardware é considerado um equipamento de computador utilizado para realizar a entrada, processamento e saída de dados. O Hardware é a parte física de um computador, formado por componentes eletrônicos, como circuitos, placas e fios, consiste em qualquer material tangível que seja necessário para fazer o computador funcionar.

1.2.1 Conceitos e definições

De acordo com Potter, Rainer E Turban (2005, p. 41) “hardware é um conjunto de dispositivos (por exemplo, processador, monitor, teclado e impressora) que, juntos, aceitam dados e informações, os processam e os apresentam”.

Segundo Mendoça e Zelenovsky (2006) os computadores proporcionam um grande impacto na sociedade. Desde os estudantes e professores até as donas de casa, dos diretores de empresas a cientistas. Quando o primeiro chip de processador foi fabricado em 1971, não se tinha em vista a revolução que começaria a partir dali. O mercado de computadores e processadores surpreendeu o mundo, provocando acontecimentos e fatos marcantes, tornando empresas pequenas em grandes empresas multinacionais. Nesse contexto entende-se a importância dos hardwares, os quais constituem toda a parte física dos computadores e ao ser combinada com os softwares, parte lógica e intangível, permitem o funcionamento dos computadores.

Um importante ponto de vista refere-se ao hardware como parte tangível do equipamento, ou seja, a um conjunto de componentes elétricos e eletrônicos que constituem um sistema de processamento de dados (KEEN, 1996).

De acordo com o site significados (2015) hardware é a parte física de um computador. É formado pelos componentes eletrônicos, circuitos de fios e luz,

placas, utensílios, unidades de processamentos e qualquer outro material em estado físico, que seja necessário para fazer com que computador funcione. O hardware é basicamente utilizado por computadores e outros equipamentos eletrônicos. Quaisquer equipamentos físicos como chaves, fechaduras, correntes e peças do próprio computador podem ser definidos como hardware. Ainda de acordo com o site, o hardware não se limita apenas a computadores pessoais, mas também está disponível em outros equipamentos como automóveis, celulares entre outros. Existem diversos tipos de hardware, que têm diferentes objetivos e funções. O hardware de rede, por exemplo, é utilizado com o propósito de possibilitar e gerir equipamentos que estão conectados em rede. Não apenas os componentes externos, como também os que estão dentro da CPU (Central Processing Unit) - Unidade Central de Processamento são classificados como hardwares, tais como a placa de memória RAM (Random Access Memory), processador, disco rígido, leitor de CD e DVD.

Outra definição interessante afirma que “O hardware, material ou ferramental é a parte física do computador, ou seja, é o conjunto de componentes eletrônicos, circuitos integrados e placas, que se comunicam através de barramentos” (www.oficinadanet, 2005).

Reforçando esses conceitos o dicionário Aurélio (2001) define hardware como componente ou o conjunto dos equipamentos físicos que compõem um computador (dispositivos de reprodução de mídias, monitor, placas, teclado, processador), juntamente com seus equipamentos periféricos.

1.2.2 Tipos de hardware

Segundo Reynolds e Stair (2011) Hardware pode ser dividido em três categorias. Os equipamentos de entrada, os quais incluem teclados, mouses, microfones ou outros equipamentos automáticos de varreduras e equipamentos que permitam ler tinta magnética. Os equipamentos de processamento incluem chips de

computador que contém a CPU e a memória principal, que são utilizados para realizar o processamento de dados. Os muitos tipos de equipamentos de saída incluem impressoras, monitores e fones de ouvidos; e têm como função trazer os dados e informações processados na CPU para o mundo real por meio de sons, imagens, vídeos e textos.

É importante ressaltar que existem diversos tipos de hardware, que podem ser divididos em três grandes categorias, conforme já definido. Neste projeto serão abordados os principais componentes de hardware.

De acordo com o site da Universidade Federal do Pará (2014), a placa-mãe é a principal placa do computador, a qual possui um conjunto de circuitos integrados que tem a função de reconhecer e gerenciar o funcionamento de todo o equipamento. A placa-mãe pode ser comparada a espinha dorsal, pois é por meio dela que o processador, que é comparado ao cérebro, estabelece contato com os demais periféricos.

Seguindo o raciocínio o site relata que a placa-mãe é responsável por interligar todos os dispositivos do equipamento, possuindo vários tipos de conectores, assim o processador, disco rígido, placa de vídeo, placa de som e placa de redes, assim como a fonte de alimentação e as memórias são instaladas em seus diferentes conectores. Toda placa-mãe possui o programa de controle BIOS (Basic Input Output System) – Sistema Básico de Entrada/Saída, armazenado em memória ROM (Read Only Memory) – Memória Somente de Leitura, que é o responsável pelo teste inicial do sistema e que guarda as configurações do hardware e as informações referentes à data e hora. Na placa mãe fica o controlador IDE (Integrated Driver Eletronics) – Controlador Eletrônico Integrado, que controla os periféricos acopladas ao microcomputador e gerencia os dispositivos de entrada e saída, como mouse, impressora e o interfaceamento USB (Universal Serial Bus) – Barramento Serial Universal, um tipo de conector universal que suporta a conexão de diversificados tipos de acessórios. As placas mãe se diferem uma da outra pelo formato, pela tecnologia suportada e pela velocidade de comunicação com os periféricos.

A memória RAM designa um tipo de memória cujos endereços podem ser acessados de modo aleatório. Surgiu em tempos que existiam somente memórias com acessos obrigatoriamente sequenciais e tem como finalidade armazenar informações e dados temporariamente para processamento do computador. A memória RAM tem duas sub-categorias, DRAM (Dinamic Random Acess Memory) - Memória Dinâmica de Acesso Aleatório, a qual é uma memória dinâmica, com baixo consumo de energia, pequeno coeficiente de aquecimento, baixo custo, porém uma velocidade menor e necessidade de atualizações. Já a memória SRAM (Static Random Acess Memory) - Memória Estática de Acesso Aleatório, é uma memória mais veloz, sem necessidade de atualização, no entanto mais cara e com um consumo maior de energia e consequentemente maior geração de calor (MENDOÇA; ZELENOVSKY 2006, p. 214).

A unidade de processamento, processador, ou CPU fica acoplada na placa-mãe. A CPU é composta por uma unidade de aritmética e lógica (ULA), que é a central do processador, onde se executam as operações aritméticas e lógicas entre dois números. Uma unidade de controle (UC), que é a unidade que armazena a posição de memória e que contém a instrução que o computador está executando nesse momento. A UC informa à ULA qual operação a executar, buscando a informação (da memória) que a ULA precisa para executá-la. Depois, transfere o resultado de volta para o local apropriado da memória. A seguir, a unidade de controle vai para a próxima instrução e uma memória central (www.ufpa.br. 2014).

A CPU é considerada a parte mais importante de um computador, pois é responsável pelo processamento de todos os tipos de dados e pela apresentação do resultado do processamento, ou seja, é a parte mais importante do computador, pois é ali onde são interpretadas e executadas as instruções fornecidas pelos aplicativos (softwares). A CPU pode ser considerada como o cérebro do computador.

Ainda segundo esse o site, o processador tem três funções básicas:

1) Realizar cálculos de operações aritméticas e comparações lógicas
2) Manter o funcionamento de todos os equipamentos e programas

3) Administrar na memória central o programa submetido e os dados transferidos de um componente ao outro, visando o seu processamento.

Para Mendonça e Zelenevosky (2006, p. 643) teclado,

> Ao contrário do que a maioria dos programadores de microcomputadores imagina, o teclado é uma via bidirecional de dados. Contudo na quase totalidade das aplicações, ele é basicamente utilizado com periférico de entrada de dados. Como Said, permite a comunicação da CPU com o processador dedicado 8048 (ou equivalente) do teclado, informando, por exemplo, os estados, aceso ou apagado de algumas teclas modificadoras, como num lock, caps lock e scroll lock, que são vistos pelos usuários através de seus respectivos leds.

De acordo com a tecla pressionada é gerado um código de varredura, que é enviado para a placa-mãe, que reconstrói o dado e o retransmite como sinal para o processador, que em contato com o software interpreta esses dados de acordo com seu contexto.

Outro hardware imprescindível para o funcionamento de um computador é o mouse, dentro das gerações mais recentes de ambientes operacionais o controle do computador é realizado apontando e clicando imagens, dispensando assim a necessidade de digitar seguidos comandos. Conforme são realizados os movimentos com o mouse o feixe de luz (laser) ou a bola, nos equipamentos mais antigos os sinais obtidos são enviados do computador para o programa, que converte o número, a combinação e a frequência dos sinais. Quando são pressionados os botões da parte superior do mouse, faz com que seja enviado um sinal ao computador, que retransmite para o programa. Baseado em quantas vezes você clica e na posição do cursor no momento do clique o programa executa a tarefa para qual foi projetado (www.filoczar.com.br. 2015).

Conforme o site UOL (2015), o dispositivo de exibição é o dispositivo de saída mais utilizado de um computador, pode ser considerado o principal hardware de saída. Quando ele é colocado em um gabinete separado, é chamado de monitor. O monitor fornece retorno instantâneo ao exibir texto e gráficos quando se trabalha ou joga. A maioria dos monitores de computador de mesa utiliza tecnologia de tela

de cristal líquido (LCD) ou tubo de raios catódicos (CRT). O Monitor se conecta ao computador por meio da placa de vídeo, gerando a interface gráfica de saída do computador. Os monitores possuem grandes variações em suas características: tipos de tela, tamanho, resolução, definição que contribuem com o modo como são exibidos os programas, vídeos e jogos utilizados no computador.

O HD (Hard Disk) – Disco Rígido, o qual é constituído por um conjunto de pratos cobertos com uma pintura magnetizável, arranjados de forma que seus centros estejam alinhados. Cada prato é logicamente subdivido em circuitos concêntricos, denominadas trilhas, onde as informações são armazenas. O transdutor responsável por ler e escrever dados é a cabeça magnética. Os pratos giram em uma rotação constante, entre 4000 e 7200 rpm. Quanto maior esta velocidade de giro, maior a capacidade para leitura e gravação de dados. O disco rígido permite que diversos tipos de arquivos, softwares, textos, imagens, vídeos e documentos de texto sejam guardados no computador e posteriormente consultados, editados e até excluídos (MENDOÇA; ZELENOVSKY, 2006).

O USB permite uma expansão externa do PC, com o USB, os usuários aproveitam os benefícios da arquitetura *Plug and Play*, ou seja, não existe a necessidade de efetuar adicionais configurações de hardware. O USB utiliza um conector universal que permite ao usuário instalar e remover periféricos sem sequer abrir o computador, além de possuir a característica de inserção e remoção automáticas, ou seja os computadores podem ser instalados e removidos em qualquer momento, mesmo que o computador esteja ligado. Dois importantes atributos do USB são também destacados: a compatibilidade universal e a simplicidade no projeto de periféricos. Esse barramento pode ser usado como a maioria dos periféricos dos computadores, como controladores de vídeo, drives de CD e DVD, unidades de fita, drives externos, máquinas fotográficas digitais, impressoras, entre outros equipamentos. Mais um aspecto importante do USB é que ele permite o compartilhamento de periféricos entre PCs, como dispositivos de segurança, telefones, monitores e outros (MENDOÇA; ZELENOVSKY 2006, p. 777).

Para concluir essa seção é importante salientar que existem outros tipos de hardwares, que não foram abordados, tais como: driver de disquete, driver reprodutor de CD e DVD, impressora, Placa PCI-Express, entre outros, entretanto devido a esse projeto não ser direcionado a hardware foram apresentados somente os tipos de hardware considerados como principais.

1.3 Software

Segundo Laudon e Laudon (2005) para desempenhar um papel útil na infraestrutura de tecnologia de informação da empresa, o hardware requer um software.

1.3.1 Conceitos e definições

De acordo com Laudon e Laudon (2005) software foi definido como: instruções detalhadas que controlam a operação de um sistema de computação.

Conforme Reynolds e Stair (2011) o software consiste em programas que comandam a operação do computador. Esses programas permitem que o computador processe as folhas de pagamento, envie contas para os clientes e forneça aos gerentes informações para aumentar os lucros, reduzir custos e oferecer o melhor serviço ao consumidor. Com o software, as pessoas podem trabalhar a qualquer hora em qualquer lugar. O software que controla as ferramentas de produção, por exemplo, pode ser utilizado para fabricar peças em quase todos os lugares do mundo.

De acordo com Potter, Rainer e Turban (2005), seguindo este conceito, a importância do software do computador não pode ser subestimada. As primeiras aplicações comerciais do computador ocorreram no início da década de 1950. Na ocasião o software era menos importante (e menos custoso) nos sistemas de computadores porque os primeiros itens de hardware literalmente tinham uma fiação

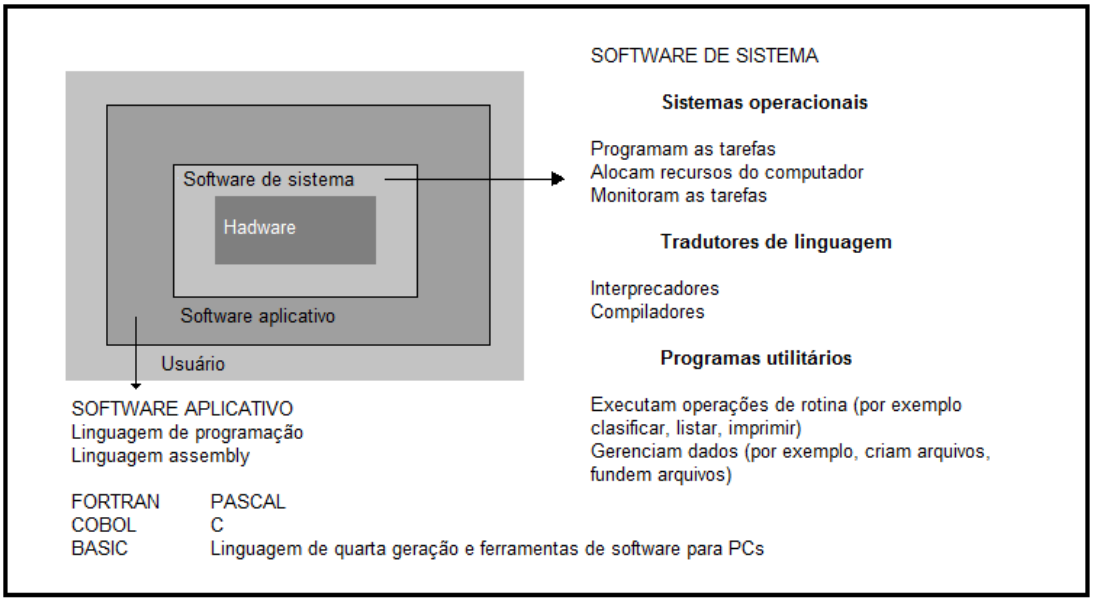
ligada manualmente para cada aplicação. Em 2005, aproximadamente, o software abrangia uma porcentagem maior dos custos de sistemas de computadores modernos, em relação ao que ocorria naquela época. O preço do hardware diminuiu enquanto a complexidade, e consequentemente o preço do software aumentou de maneira considerável.

Potter, Rainer e Turban (2005) afirmam que a maior complexidade de software também aumentou o potencial para erros, sendo assim, grandes aplicações podem conter milhões de linhas de códigos de computador, escrito por centena de pessoas no decorrer de vários anos. Independente das tendências gerais no software (maior complexidade, maior custo, maiores quantidades de defeitos, maior uso do software aberto), o software é onipresente em nossas vidas profissionais e pessoais independente do setor em que se trabalha, estará sempre rodeado por algum tipo de software. Reforçando este conceito, os autores afirmam que software consiste em programas de computador que são sequencias de instruções para o computador, e, ao contrário dos primeiros computadores da década de 1950, o software moderno utiliza o conceito de programa armazenados, em que os programas de software armazenados são acessados e suas instruções são executadas na CPU do computador, assim que um programa termina sua execução, um novo programa é carregado na memória principal.

1.3.2 Tipos de software

Laudon e Laudon (2005) explicam que os softwares são classificados em dois tipos: o software de sistema e o software de aplicativo. Eles são inter-relacionados e podem ser vistos como um conjunto de caixas uma dentro da outra, sendo que, cada uma delas deve interagir com as outras caixas em sua volta conforme mostra a figura 1.1.

Figura 1.1- Relacionamento dos tipos de software



Fonte: adaptado de LAUDON; LAUDON, 2005, p. 197.

Segundo os autores o software de sistemas é um conjunto de instruções que atua basicamente como intermediário entre hardware e software e os programas e aplicativos, e também pode ser manipulado diretamente pelos usuários experientes. O software de sistemas fornece funções importantes de autocontrole para os sistemas do computador como carregamento automático durante o processo de inicialização do computador, gerenciamento de recursos de hardware, como armazenamento secundário para todos os aplicativos, e conjuntos de instruções mais usados, para serem utilizados por todos os aplicativos. Os autores explicam ainda que software de sistemas é a classe de programas que controla e dá suporte ao sistema do computador e às respectivas atividades de processamento de informações. O software de sistemas também facilita a programação, teste e depuração os programas de computador (LAUDON; LAUDON, 2005).

Para Potter, Rainer e Turban (2005) o software de sistemas pode ser agrupado em duas categorias funcionais: programas de controle do sistema, que controlam o uso de recursos do hardware e dos dados de um sistema de computador e programas para suporte dos sistemas e programas de suporte de sistemas, suportam a operação, o gerenciamento e os usuários do computador, oferecendo diversos serviços de suporte.

Seguindo o raciocínio, “O software aplicativo consiste em instruções que orientam o computador a executar atividades de processamento de informações específicas e que oferecem grupos de funções para os usuários” (POTTER; RAINER; TURBAN, 2005, p. 498).

Paulino (2009) aborda de forma não menos importante, que existem mais oito tipos de softwares aplicativos que devem ser considerados:

1) Linguagem de programação: Tem como finalidade o desenvolvimento de outros programas de uso genérico.

2) Software como serviço: Saas, é um modelo de distribuição de software, no qual não é vendido e instalado localmente, mas sim é liberado apenas o acesso ao serviço oferecido por este software e é licenciado para utilização através da internet.

3) Softwares tutoriais: Geralmente usados para informar ou ensinar sobre determinado assunto, muito usados em treinamentos.

4) Software de exercitação: Similar ao software tutorial, porém o usuário conta com maior interatividade através de resposta diante de questões apresentadas.

5) Software de investigação: Nesta categoria se enquadra todo os softwares que permitem utilizar a localização de diversas informações a respeito de diversos assuntos.

6) Software de simulação: Geralmente utilizados para simulações de situações da vida real. Dentre os mais conhecidos estão os simuladores de voo e gerenciadores de cidades, utilizados para situações de treinamentos de pessoas para enfrentar casos do dia a dia.

7) Software de jogos: Geralmente relacionados a entretenimento para proporcionar lazer e diversão.

8) Softwares abertos: São os que permitem que o usuário produza com liberdade e criatividade, se classificam nessa categoria os softwares de banco de dados.

1.4 Engenharia de Software

Segundo Pressman (2011, pg. 48), “a engenharia de software engloba processos, métodos e ferramentas que possibilitam a construção de sistemas complexos baseados em computador dentro do prazo e com qualidade”. Para o autor o processo de software “incorpora cinco atividades estruturais: comunicação, planejamento, modelagem, construção e emprego; e elas se aplicam a todos os projetos de software”.

Tonsig (2003) define que grande parte dos objetos utilizados atualmente, são construídos com o emprego de componentes, tais componentes são partes que, uma vez agregados convenientemente, geram uma arquitetura utilizável com uma aplicação.

1.4.1 Conceitos e Definições

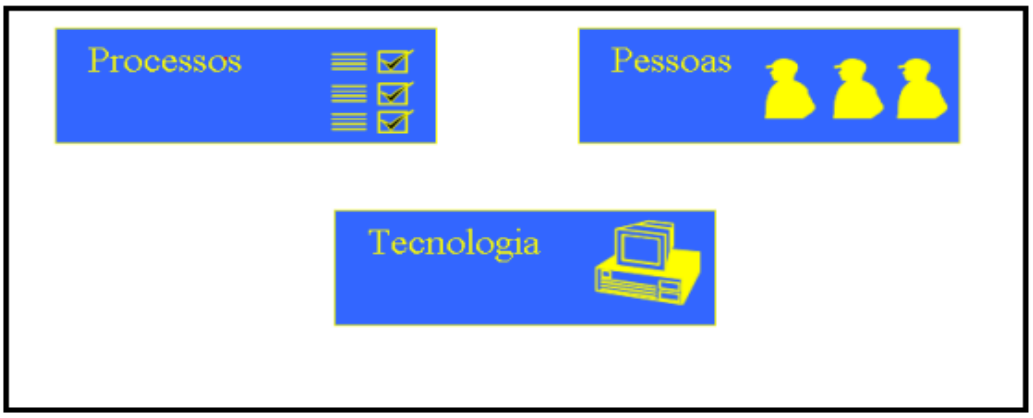
A partir do início da década de 1960, aproximadamente em 1961, começaram a surgir computadores mais modernos, e assim, com esses computadores que tinham disponibilidade para um maior poder computacional, o software ganhou grande destaque e, por conta disso, uma série de problemas relacionados à falta de conhecimento com o qual sua construção era realizada ficou indubitável, apresentando o cenário da época como uma situação caótica.

Devido a esta situação, iniciou-se, em meados de 1968, uma fase que foi chamada de Crise do Software. Esse termo era utilizado para expressar as dificuldades para o desenvolvimento de um software frente ao rápido crescimento da demanda, da inexistência de técnicas estabelecidas para o desenvolvimento de sistemas que funcionassem adequadamente ou que pudessem ser validados de alguma forma qualquer, e da complexidade dos softwares que tinham problemas a serem resolvidos. Assim deve-se ser levado em consideração que um sistema de software complexo é caracterizado por componentes abstratos de software, que são estruturas de dados e algoritmos, utilizados na forma de procedimentos,

funções, módulos, objetos ou agentes, que compõem a arquitetura do software, que deverão ser executados em sistemas computacionais (TONSIG, 2003).

Conforme Santos Neto (2007), a Engenharia de Software tem como base a tecnologia, as pessoas e os processos, como mostra a figura 1.2, e é preciso um equilíbrio entre eles para que o software seja bem construído. Sendo assim, o software precisa de um planejamento detalhado para que seu desenvolvimento seja realizado de maneira competente, fazendo com que os riscos sejam pequenos.

Figura 1.2- Tripé da Engenharia de software



Fonte: SANTOS NETO, 2007, p. 23.

Devido à falta de conhecimento em como desenvolver um software e visando discutir alternativas para contornar a Crise do Software, em 1968 foi criado um evento com esse objetivo, a *North Atlantic Treaty Organization (NATO) Software Engineering Conference*[3]. E com essa conferência foi marcado o início dessa nova área da computação (SANTOS NETO, 2007).

Friedrich Ludwig Bauer[4] foi o primeiro a definir a Engenharia de Software, em 1968 na conferência NATO[3], segundo ele "Engenharia de Software é a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe em máquinas reais". Devendo-se levar

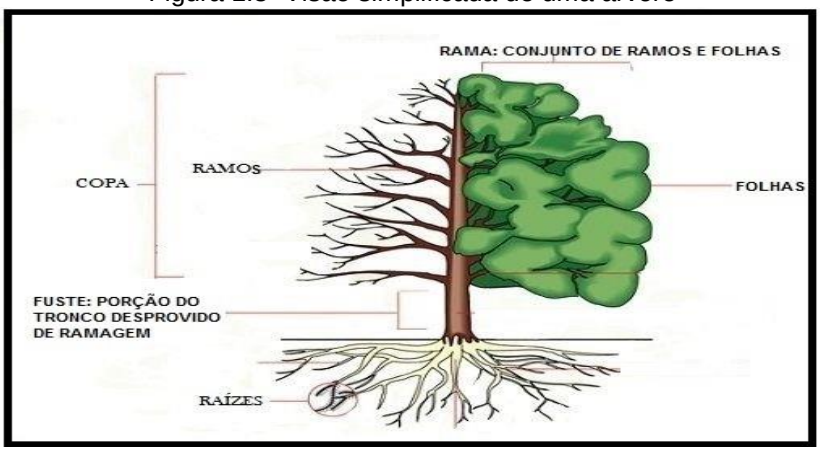
[3] **NATO SOFTWARE ENGINEERING CONFERENCE** foi uma conferência internacional que aconteceu em Garmish, Alemanha em 1968 e 1969 (SANTOS NETO, 2007.).

[4] **FRIEDRICH LUDWIG BAUER** (Ratisbona, 10 de Junho de 1924 — 26 de Março de 2015) foi um cientista da computação alemão e professor emérito na Universidade Técnica de Munique (SANTOS NETO, 2007.).

em consideração que os conceitos de criação, construção, análise, desenvolvimento e manutenção já são englobados no próprio significado de engenharia.

Segundo Tonsig (2003, p. 5, grifo do autor) “[...] pode-se tentar fazer uma primeira generalização acerca de sistemas: *todos os sistemas possuem, explícito, um objetivo declarado.* Não significa que não possa vir a ter outros objetivos, ocultos ou correlatos ao explícito” e ainda que “[...] sistemas são compostos por partes (entidades), cada qual com uma função (especialização); juntas permitem o perfeito funcionamento do sistema. Vamos focar a árvore e checar as partes que a constitui”, conforme pode ser observado na figura 1.3.

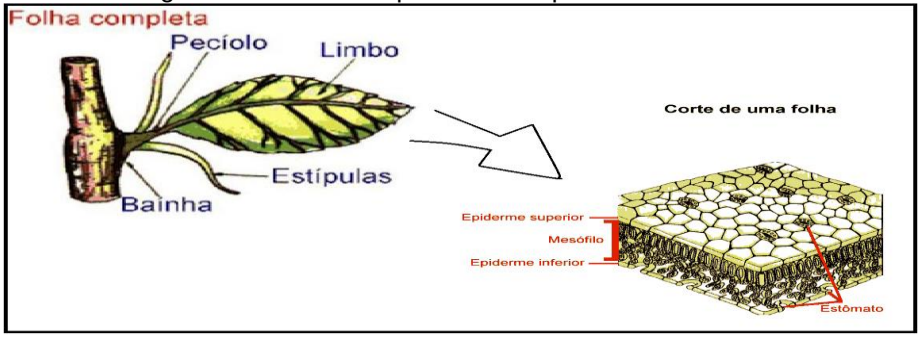
Figura 1.3- Visão simplificada de uma árvore



Fonte: adaptado de TONSIG, 2003, p.7

“Cada uma dessas partes, por sua vez, é constituída de outras partes (subsistemas). Veja, por exemplo, a estrutura simplificada de uma folha [...]” (TONSIG, 2003, p. 7) conforme demonstrado na figura 1.4.

Figura 1.4– Visão simplificada das partes de uma folha



Fonte: adaptado de TONSIG, 2003, p.7.

De acordo com Tonsig (2003, p. 54), pode-se afirmar que software “[...] não é uma entidade corpórea, trata-se de uma entidade lógica e não física; portanto, não se desgasta”, pois segundo o autor, o software pode ter as suas finalidades alteradas, fazendo com que o software não atenda mais aos requisitos ou quesitos atuais, pela mudança no cenário que ele atendia, e não pelo desgaste do tempo. Mas para a Engenharia de Software, ele deve ser visto como um produto, algo a ser vendido. Deve-se levar em consideração que quem o utilizará será o cliente, que necessariamente, é alguém diferente do programador.

Segundo Alvarez (1990 apud TONSIG, 2003), deve-se utilizar métodos para o desenvolvimento do software, o objetivo básico para isso é obter uma consistência maior no trabalho, maior qualidade ao usuário, diminuir perdas que podem ser acarretadas por caminhos sem saída, facilidade no treinamento de novos analistas, e um controle maior no desenvolvimento. Sendo assim, Pressman (1995) explica que:

> Os métodos envolvem um amplo conjunto de tarefas que incluem: planejamento e estimativa de projeto, análise de requisitos de software e de sistemas, projeto de estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção (PRESSMAN 1995 apud TONSIG, 2003, p. 54).

Qualquer método escolhido para o desenvolvimento de um software será formado por um ciclo de vida de desenvolvimento, que pode ser visto como um roteiro para o trabalho, constituído normalmente por macro etapas com os objetivos funcionais para a construção de um software. Normalmente, ao desenvolver um software, qualquer que seja o modelo utilizado, compreende três fases: requisitos, projeto e desenvolvimento, implementação e manutenção (TONSIG, 2003).

Para conseguir desenvolver um software capaz de satisfazer às necessidades de seus usuários, com qualidade que perdure, por intermédio de uma arquitetura sólida que aceite modificações, de forma rápida, eficiente, com o mínimo de desperdício e retrabalho, é indispensável o emprego de modelagem que é a parte central de todas as atividades que levam à implementação de um bom software (RUMBAUGH, 2000 apud TONSIG, 2003). Reforçando essa ideia, Pressman (2011)

aponta que “A prática de engenharia de software é uma atividade de resolução de problemas que segue um conjunto de princípios básicos”.

1.5 Qualidade de Software

De acordo com Colombo e Guerra (2009) qualidade é a conformidade dos requisitos, e estes devem estar definidos para permitir que sejam gerenciados com o uso de medidas, de forma a reduzir o retrabalho e aumentar a produtividade. Ainda de acordo com os autores em uma breve história do desenvolvimento de software relata que os problemas e a busca pela qualidade dos produtos são uma das preocupações das organizações. Os softwares, durante anos, tiveram uma evolução muito rápida e concentrada, mas que não acompanhavam a evolução exponencial dos hardwares, e por este motivo foi verificado a necessidade de se fazer software cada vez com mais rapidez, ocasionando vários erros.

“A ideia de qualidade é aparentemente intuitiva; contudo, quando examinado mais longamente, o conceito se revela complexo. Definir qualidade para estabelecer objetivos é, assim, uma tarefa menos trivial do que aparenta a princípio” (KOSCIANSKI; SOARES, 2007, p. 17). Os autores esclarecem que a qualidade de software está ligada a diversos fatores, entre eles: as pessoas envolvidas, o tempo para desenvolvimento, o orçamento, disponível, os requisitos solicitados, e vários outros, estes fatores que influenciam diretamente na qualidade podem ter seus erros minimizados através de parâmetros introduzidos com as certificações e testes, os quais foram sendo aperfeiçoados com o passar do tempo, Mas ainda não pode se afirmar que um projeto de software será perfeito, sem nenhuma falha.

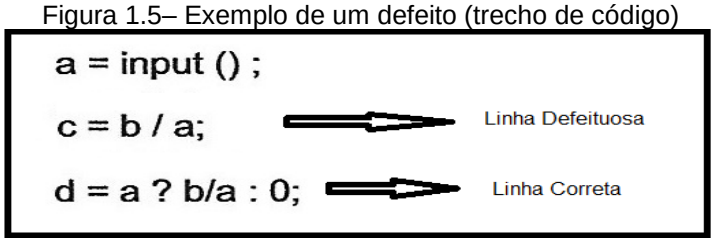
Koscianski e Soares (2007) avaliam, que com as normatizações ou certificações, as falhas ou até os defeitos serão minimizados, mas não eliminados, porque a construção de um novo software é mais complexa do que, por exemplo, a complexidade na diferença entre construir uma residência térrea e um prédio de dez andares, neste exemplo, existe, com certeza, uma diferença de cálculos necessários entre os dois tipos de construção, mas outros fatores são iguais: o tijolo, cimento,

pedra, ferro são materiais iguais, que não são alterados, pelo tipo de construção, já no desenvolvimento de um software há várias interações entre os diversos componentes do sistema e de diversas fontes que não se tem o controle de como vai se integrar ao software e que são obrigados a funcionarem em harmonia e sincronismo.

Koscianski e Soares (2007) continuam explicando que é necessário enfatizar a diferença entre defeito e falha, embora as das palavras tragam ideias parecidas, não são sinônimos entre si e são usadas para conceitos distintos, para os autores, defeito é uma imperfeição de um produto, pois o defeito faz parte do produto e, em geral, refere-se a algo que está implementado no código de maneira incorreta.

Na figura 1.5, é apresentado um trecho de código com defeito.

Figura 1.5– Exemplo de um defeito (trecho de código)

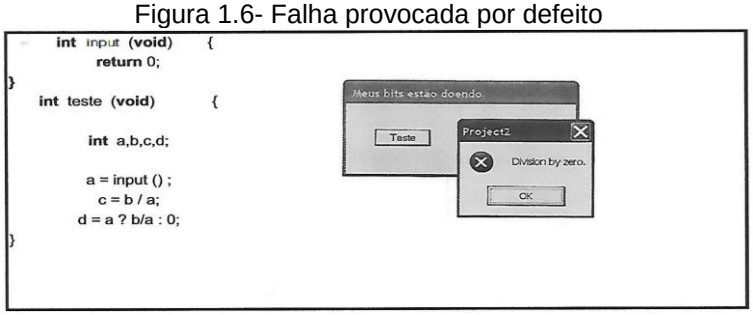


Fonte: adaptado de KOSCIANSKI e SOARES, 2007, p.32.

No trecho do código demonstrado na figura 1.5 a expressão que calcula o valor para a variável c ‘pode’ apresentar um problema, se for atribuído a ela um valor nulo, a operação de divisão por zero provocará algo anormal na execução do programa, já na linha seguinte onde uma atribuição é feita à variável d, este problema não acontece.

“Falha é o resultado errado provocado por um defeito ou condição inesperada” (KOSCIANSKI e SOARES, 2007, p. 32). Ao executar o programa a linha defeituosa causa uma parada quando se tenta fazer a divisão por zero, isto é, aparece a imperfeição feita durante a construção do software como mostra a figura 1.6.

Figura 1.6- Falha provocada por defeito



Fonte: KOSCIANSKI e SOARES, 2007, p.32.

Koscianski e Soares (2007) explicam que podem existir outros tipos de defeitos que não interrompa a execução do programa, estes erros podem não ser aparentes mais pode ser que ocasione uma falha muito grave.

Outro exemplo que ilustra muito bem se a qualidade de software não for feita de maneira sistemática e efetuados tantos testes forem necessários, foi o que aconteceu, fartamente reportado na internet, com a explosão logo após o lançamento do foguete ARIANE 501, em 4 de junho de 1996 na Guiana Francesa. O mesmo se desviou de sua rota e ocorreu a autodestruição, depois de feitas as investigações foi divulgado um relatório informando que ocorreu um erro numérico (overflow) em uma linha de código no programa do foguete; em um cálculo que tinha números em ponto flutuantes representados por 64 bits, este número foi convertido em um inteiro de 16 bits, e a falha deu-se devido ao número ser muito grande para ser representado em 16 bits, tudo porque devido ao pouco tempo necessário para o lançamento, o programador copiou e fez algumas adaptações do código original da série de foguetes ARIANE 4, e esta linha que tinha erro passou sem ter sido testada, outras linhas tinham o mesmo defeito mais foi verificado e corrigido, fato que não foi percebido nesta linha em específico, causando uma falha no cálculo da trajetória do foguete devido à um erro de codificação (www.ime.uerj.br, 1996).

Existem ainda diversos outros fatos relacionados a erros. Na Internet encontra-se uma infindável quantidade de exemplos de falhas devido a defeitos de programação nos softwares; defeitos estes que em sua maioria poderiam ser detectados se fossem seguidos padrões criteriosos de processos e testes (KOSCIANSKI e SOARES, 2007).

Segundo Bartié (2002) qualquer alteração durante o processo de desenvolvimento de um software poderá comprometer a qualidade final deste produto, pois a qualidade final do produto é exatamente a soma de todas as decisões e realizações ocorridas durante o processo de desenvolvimento. Se a intenção for a de se produzir softwares com alta qualidade será necessário o investimento em qualidade em todos os pontos do processo. Ainda segundo o autor, softwares mal testados podem provocar grandes prejuízos às organizações. Um simples erro interno do produto poderá afetar as decisões dos gerentes, diretores em ter outros, pois estão apoiados na qualidade das informações consultadas, e em caso de erros poderiam autorizar compras desnecessárias, solicitar manutenção antes ou depois do ideal, produzir estatistas irreais de produtividade, entre outros. No entanto não é possível obter-se qualidade de software onde não há o enfoque simultâneo em produto tecnológico e processo de desenvolvimento deste produto.

Com base nas informações apresentadas, para o autor, para que o produto seja de qualidade, é necessário que tenha qualidade em todas as fases, das quais se destacam as seis consideradas principais:

1) Requisitos dos usuários;
2) Requisitos técnicos;
3) Implementação da programação;
4) Testes para busca de erros e falhas;
5) Aceitação pelo usuário e
6) Implantação e treinamento ao usuário.

Para Bartié (2002), estas fases são necessárias, pois o usuário quer qualidade e para ele, softwares de qualidade garantem que haja segurança nas transações, nos negócios, das pessoas envolvidas e mantém a alta disponibilidade dos serviços. Para aferir esta qualidade é necessário, segundo os autores três passos principais:

1º) A garantia que se atinge seguindo padrões que garantam a qualidade do software;

2º) O planejamento através da seleção de procedimentos e padrões adequados ao projeto;

3º) O controle que é assegurando se no desenvolvimento estão sendo seguidos os procedimentos e padrões de qualidade do projeto.

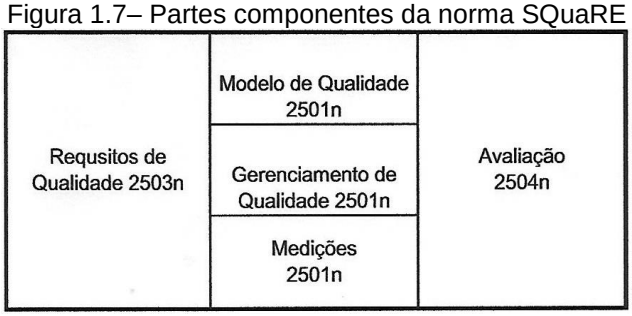
Resumindo, “Qualidade de Software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos” (BARTIÉ, 2002, p.16).

Conforme foi abordado até o momento, para se garantir a qualidade do software, são necessários vários procedimentos na sua construção. Bartié (2002) comenta que as organizações trabalham com o conceito de ‘Zero-defeito’, que é não tolerância a erros dentro do processo de construção do software, o objetivo é definir todo o processo para minimizar a incidência de problemas. Desta maneira reduzindo os defeitos e consequentemente as falhas, mas também, o que se torna o principal motivo para as organizações, é que os custos diminuem fazendo que a empresa consiga obter vantagens sobre a concorrência e o cliente além da parte financeira direta, conseguirá o retorno através da diminuição dos custos relativos tempo dispendido com manutenções e correções. “O ‘legal’ sobre os padrões é que existem muitos deles para escolher” (TANENBAUM, s.d. apud KOSCIANSKI e SOARES, 2007, p. 204).

Os autores pesquisados apresentam a necessidade de padronização na execução dos projetos, trazendo assim maior confiabilidade e qualidade do software, e para isto foram criados, por entidades internacionais e nacionais, alguns padrões.

Segundo Koscianski e Soares (2007), um dos padrões atualmente mais aceitos é a norma internacional ISO/IEC 25010:2011 – SquaRE (Product Quality Requirements and Evaluation), este projeto que reorganizou o material de séries já existentes, adotou uma nova divisão como mostra a figura 1.7:

Figura 1.7– Partes componentes da norma SQuaRE

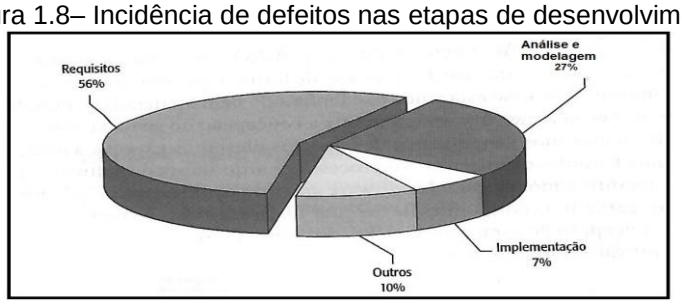


Fonte: KOSCIANSKI e SOARES, 2007, p. 207.

Considere que o cliente indique que a usabilidade seja um aspecto importante do software. Essa característica pode estar relacionada com muitos outros fatores diferentes: o cliente poderia afirmar, por exemplo, que se o programa demora um tempo muito longo para responder torna-se difícil de usar. Embora isto possa ser verdade, ao verificar os requisitos do cliente para o modelo 9126, o tempo de resposta aparece apenas na subcaracterística de eficiência. Ele pode se tornar um problema de usabilidade caso o usuário não consiga controlar ou operar o software – o que corresponde precisamente à definição da subcaracterística de operabilidade (KOSCIANSKI e SOARES, 2007).

Existe uma visão geral de que os defeitos estão somente no código-fonte do produto. Também se tem o entendimento que os profissionais de desenvolvimento, qualidade e teste são os únicos responsáveis pelo software de qualidade, mas como representado na figura 1.8 os defeitos podem estar em todas as etapas do desenvolvimento (BARTIÉ, 2002).

Figura 1.8– Incidência de defeitos nas etapas de desenvolvimento

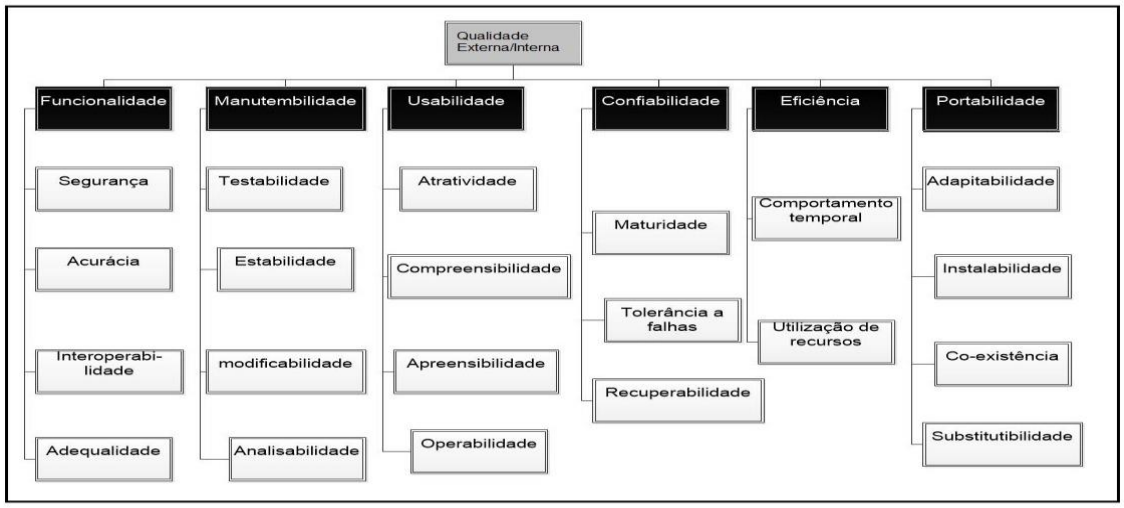


Fonte: BARTIÉ, 2002, p. 26.

O modelo SQuaRE é considerado hierárquico, como explicam Koscianski e Soares (2007), isto é, todas as características de qualidade possui uma

subcaracterística, assim se uma característica esta sendo avaliada não se consideram os fatores relativos a outra característica, ficando mais fácil de identificar e fazer as correções necessárias, este modelo segue o que havia sido decidido na norma ISO/IEC 9126 que evoluiu a ISO/IEC 25010, esta divisão hierárquica permite precisar com uma descrição muito mais rápida, os requisitos de qualidade de um software. A figura 1.9 apresenta as características e subcaracteristicas:

Figura 1.9 - Modelo de qualidade do ISO/IEC 9126



Fonte: adaptado de KOSCIANSKI e SOARES, 2007, p. 210.

Conforme explicam Koscianski e Soares (2007) a resposta da pergunta na qual se deve aplicar qualidade, torna-se simples, tem que se aplicar qualidade em todas as etapas do desenvolvimento, pois a cultura da qualidade passa a criar um ambiente favorável a prevenção e detecção de defeitos, evitando o retrabalho desnecessário, e consequente perda de tempo e recursos.

A implementação do processo completo de controle de qualidade só seria possível em mundo utópico, sem restrições, mas no mundo real envolve um grande montante de recursos: recursos financeiros; recursos humanos e tempo, onde nem sempre há condições de se obter. Desta forma os riscos são minimizados, utilizando bons profissionais que conseguem direcionar os esforços e controlar as restrições a eles impostas. Ainda no aspecto relacionado à maturidade, está o cliente que ainda não possui um processo estruturado, e com certeza não será possível fornecer

todos os dados para elaboração dos documentos de requisitos iniciais que garantam a qualidade do processo (BARTIÉ, 2002).

1.6 Desenvolvimento de Software

Pressman (2011, p. 48) considera o Software como “o elemento-chave na evolução de produtos e sistemas baseados em computador e uma das mais importantes tecnologias no cenário mundial”. E ainda complementa dizendo que no decorrer dos últimos cinquenta anos o software evoluiu, deixando de ser uma ferramenta especializada em resolução de problemas e análise de informações, para uma indústria propriamente dita.

Ainda segundo Pressman (2011, p. 48), “David Hooker [Hoo96] propôs sete princípios que se concentram na prática da engenharia de software como um todo”:

1º) A razão de existir: por que construir o software.

2º) Fazer de forma simples: construir da forma mais simples possível.

3º) Manter a visão: seguir sempre a arquitetura do projeto.

4º) O que um produz, outros consomem: quase sempre, quem utiliza o software não é quem o produz.

5º) Estar aberto ao futuro: estar sempre preparado para mudanças nas utilizações do software.

6º) Planejar com antecedência, visando a reutilização: a reutilização do código e projetos pode se tornar um grande benefício, por disponibilizar, depois de bem montados, mais tempo e menos custo.

7º) Pensar: pensar muito e de forma clara quase sempre produz resultados melhores.

Essa ideia de que o software evoluiu é reforçada por Leite (2013), que entende que o desenvolvimento de software deixou de ser sinônimo apenas de código faz anos. Atualmente, sabe-se que se deve ser levado em consideração a engenharia e a qualidade de software, e sabe-se também da necessidade da utilização de uma metodologia de trabalho. Considerando-se metodologia, como a

maneira de se utilizar um conjunto coordenado e coerente de métodos para conseguir atingir um objetivo específico, de modo a evitar, tanto quanto seja possível, a particularidade na execução de cada trabalho. Fornecendo, assim, um roteiro, ou seja, um processo dinâmico e interativo para o desenvolvimento estruturado de projetos, software ou sistemas, visando desenvolver e manter qualidade e produtividade dos projetos.

Mas, mesmo com o conhecimento de que se devem utilizar as metodologias ou roteiros, Pressman (2011) acredita que ainda existem problemas no desenvolvimento de um software que tenha boa qualidade e que respeite o prazo e os custos estabelecidos.

De acordo com Miles e Pilone (2008), essa ideia é reforçada, pois para eles o desenvolvimento começa com a necessidade do cliente, uma ideia dele, mas transformar essas ideias em requisitos dentro de um código funcional, o qual possa satisfazer essa necessidade, não é simples, pois é incomum o cliente realmente saber o que ele quer e precisa no início do projeto.

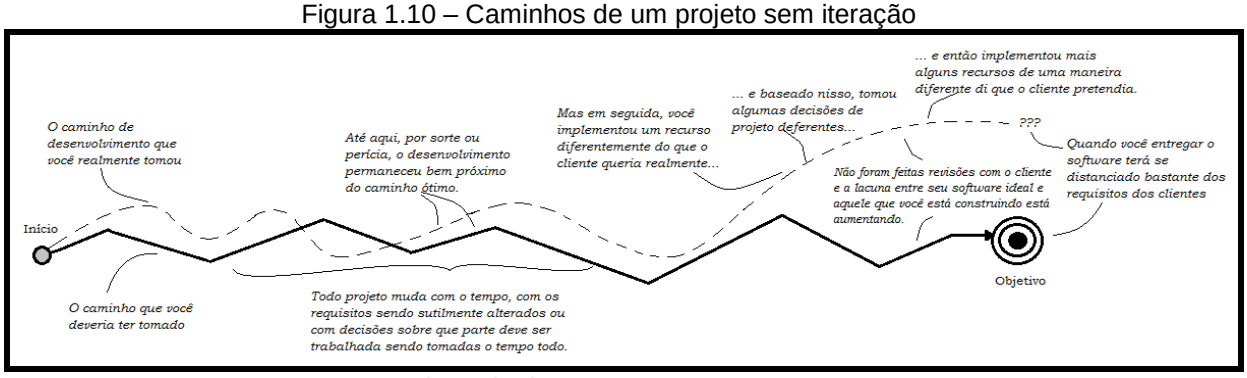
Os autores ainda afirmam que o desenvolvimento de software não é e não pode ser considerada uma especulação. Durante o desenvolvimento deve-se manter o cliente no circuito para certificar-se de que o projeto está no caminho certo. O cliente deve ser informado de qualquer alteração que seja realizada, como está sendo realizada, e qual será sua funcionalidade, o que é denominado por eles de iteração com o cliente. Por esse motivo, um requisito, para ser considerado útil, na perspectiva dos autores, deve ser escrito no ponto de vista do cliente, descrevendo detalhadamente o que o software fará, e como funcionará**.** Ou seja, um requisito tem que ser escrito para o cliente, em uma linguagem que ele entenda facilmente, pois este requisito deve poder ser lido como um roteiro de usuário, um roteiro que servirá de base para que eles saibam como interagirão com o software que está sendo construído.

O que é reforçado pela explicação e exemplos de Pressman (2011, p. 47):

> Todo projeto de software é motivado por alguma necessidade de negócios – a necessidade de corrigir um defeito em uma aplicação existente; a necessidade de adaptar um “sistema legado” a um ambiente de negócios em constante transformação; a necessidade de estender as funções e os recursos de uma aplicação existente, ou a necessidade de criar um novo produto, serviço ou sistema.

Sendo assim, conclui-se que as ideias e as necessidades do cliente devem ser a fonte para a determinação dos requisitos básicos do software a ser construído. E para os autores para se conseguir bons requisitos, também é necessário envolver o máximo de colaboradores possível, além de ter uma comunicação contínua com o cliente. Para que ao final do projeto o desenvolvimento não esteja distante do que foi planejado. A figura 1.10 mostra como os caminhos se distanciam quando o planejamento e o desenvolvimento não são acompanhados pelo cliente.

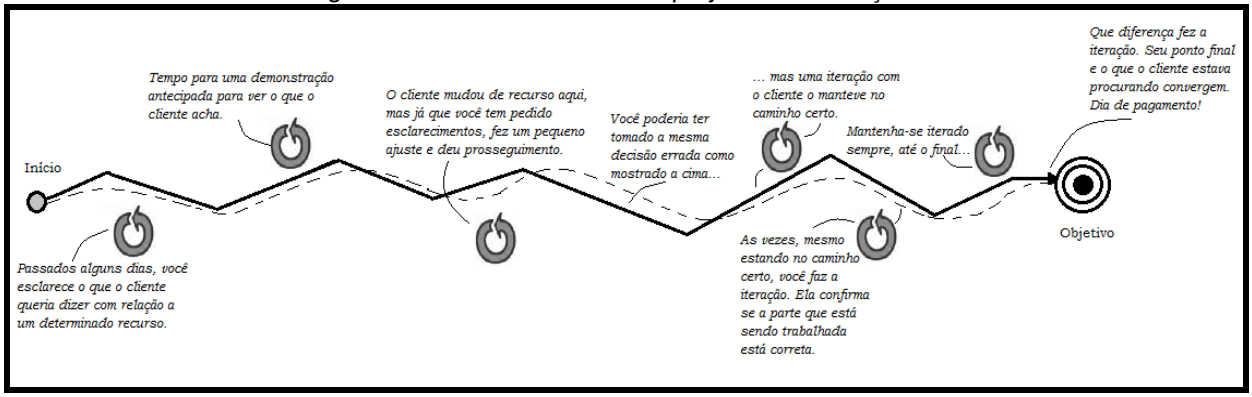
Figura 1.10 – Caminhos de um projeto sem iteração



Fonte: adaptado de MILES e PILONE, 2008, p.8.

A figura 1.11 mostra como os caminhos, mesmo que não sejam totalmente iguais, ficam muito próximos quando o planejamento e o desenvolvimento são acompanhados pelo cliente.

Figura 1.11– Caminhos de um projeto com iteração



Fonte: adaptado de MILES e PILONE, 2008, p.8.

Miles e Pilone (2008) afirmam que a iteração é como um check-up frequente em seu software. Você sempre saberá como está se saindo.

Sobre a metodologia, Leite (2013) explica que é objetivo a definição para todos os que estão envolvidos direta ou indiretamente com o desenvolvimento do software de forma clara e precisa de quem faz o que; quando; como e onde, e ainda deve definir os papéis dos técnicos, dos usuários e da administração da empresa no processo de desenvolvimento. Com isto evita que a situação de conhecimento, sobre o sistema seja de poucos, conhecidos comumente como donos do sistema, além disto, deve conter um conjunto de padrões preestabelecidos, de modo a garantir fácil integração entre os sistemas e evitando a subjetividade na abordagem. Ainda segundo o autor, o uso da uma metodologia pode possibilitar ao gerente controlar o projeto do desenvolvimento de software, mantendo o projeto sobre controle para que não ocorram desvios de planejamentos de custos e prazos, que, se de alguma forma forem negligenciados ou malconduzidos, podem colocar em risco o sucesso do projeto. E ao desenvolvedor possibilita obter uma base para que possa produzir de maneira eficiente e em menor tempo, um software de qualidade que consiga satisfazer os requisitos estabelecidos.

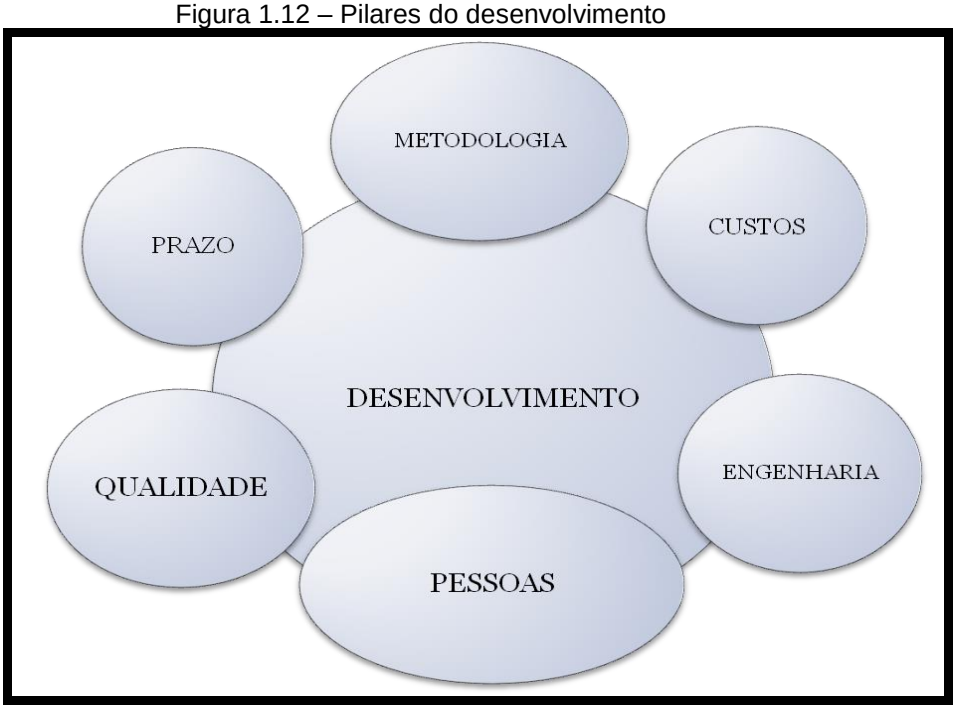
Leite (2013) esclarece que devido ao fato de uma metodologia ser um conjunto de métodos, é conveniente definir o que é método e qual é o seu objetivo. Segundo ele, um método é abordagem técnica passo a passo para realizar uma ou mais tarefas indicadas na metodologia. Ou seja, são os procedimentos que se fazem necessários ser adotados para conseguir-se atingir um objetivo. Por outro lado, uma técnica pode ser entendida como uma maneira apropriada de se investigar sistematicamente um universo de interesse ou domínio do problema.

Para isso, deve-se ser utilizada uma notação. Para exemplo de técnica, é apresentado: Análise estruturada, Análise Essencial, Projeto Estruturado, Análise Orientada a Objetos. Ainda de acordo com o autor, diversas vezes, utilizar uma metodologia pode ser encarado como cerceamento, limitação da criatividade dos técnicos, ou como, acréscimo de burocracia, ou seja, as documentações, que é tido

como desnecessários por muitos para a construção de um software. Uma metodologia não deve limitar a criatividade profissional, mas sim, ser um instrumento que deve determinar um planejamento sistemático, que alia, concilia e coordena todas as áreas envolvidas. Os requisitos de qualidade e produtividade de um projeto podem limitar a criatividade, não a metodologia. Ainda informa que, toda escolha de uma metodologia a ser utilizada no desenvolvimento, deverá ser realizada com base na natureza do projeto e do produto a ser desenvolvido, dos métodos e ferramentas a serem utilizados e também, dos controles e produtos intermediários desejados. O autor conclui que, o uso de metodologia, mesmo que não seja tão fortemente sedimentada para desenvolvimento de software, é de extrema importância, para que com um mínimo de qualidade, o sistema construído possa atender as necessidades dos clientes (LEITE, 2013).

Deve-se levar em consideração que para o desenvolvimento do software deverá ser utilizado tanto à metodologia explicada e a engenharia, quanto às características e formas de medir a qualidade do software, como é exemplificado de maneira simples na figura 1.12:

Figura 1.12 – Pilares do desenvolvimento



Fonte: Autoria própria, 2015.

1.7 Testes

De forma que se obtenham produtos mais confiáveis, seguros, com melhor desempenho e que principalmente atendam de maneira satisfatória às necessidades dos usuários é necessário que sua produção seja regida por métodos, padrões e também processos bem estruturados. Nesse sentido a área de Engenharia de Software tem profundas contribuições a fazer, bem como sua especialização e foco do projeto desenvolvido: O teste de software (FAPEG, 2013).

De acordo com o site Crowdtest (2015), teste é parte fundamental no ciclo de vida de um software. Desta forma o site lista sete princípios fundamentais que envolvem o processo de teste e devem servir como um guia geral, tanto para testadores quanto para desenvolvedores. Afinal, ambos participam efetivamente do processo de amadurecimento do sistema.

1º) Testes apontam a presença de falhas: Testes conseguem identificar a existência de falhas, mas não pode garantir a ausência delas. Mesmo se nenhum erro for identificado em uma bateria de testes, não é possível afirmar que o software está livre de falhas;

2º) Teste exaustivo é impossível: A menos que a aplicação sendo testada tenha uma estrutura lógica muito simples e valores de entrada limitados, teste exaustivo é inviável pois seria extremamente custoso cobrir todos os cenários possíveis. Deve-se calcular o esforço dos testes baseando-se nos riscos e prioridades;

3º) Teste antecipado: Ao desenvolver um software, as atividades de teste devem começar o quanto antes. Assim que os requisitos ou modelagem do sistema estiverem prontos, é possível começar o trabalho de modelagem do plano de testes. O quanto antes uma falha for identificada no ciclo de vida de um sistema, mais barata e mais simples será a correção;

4º) Agrupamento de falhas: A maioria das falhas encontradas durante a execução dos testes está concentrada em um número pequeno de módulos. Sempre existe uma área do software que é responsável pelo maior número de erros;

5º) Paradoxo do pesticida: Um conjunto de testes, se executado várias vezes, pode não mais detectar novas falhas. Para contornar esse problema, os casos de teste devem ser frequentemente revisados e atualizados. Eles devem ser reformulados para abordar novas áreas do sistema e assim aumentar a chance de detectar novas falhas;

6º) Teste depende de contexto: Os testes devem ser elaborados de acordo com o tipo do software. Por exemplo, um sistema bancário deve ser testado de maneira diferente de uma rede social. Há questões de segurança que devem ser mais precisamente abordadas no primeiro caso. Da mesma forma que testes web são elaborados com foco diferente dos testes de aplicações desktop;

7º) Ausência de erros é uma ilusão: Identificar e corrigir os problemas de um software não garantem que ele está pronto. Os testes foram elaborados para identificar todas as possíveis falhas? O sistema atende às necessidades e expectativas dos usuários? Ou seja, existem outros fatores que devem ser considerados para garantir a qualidade do sistema.

De acordo com a Comissão Internacional para Qualificação de Teste de Software (2005), os testes podem demonstrar falhas, que são causadas por defeitos, segundo alguns dos conceitos descritos no dicionário Priberam Online, falha é o ato de não produzir o efeito desejado, e defeito é um inconveniente, estorvo ou uma deformidade.

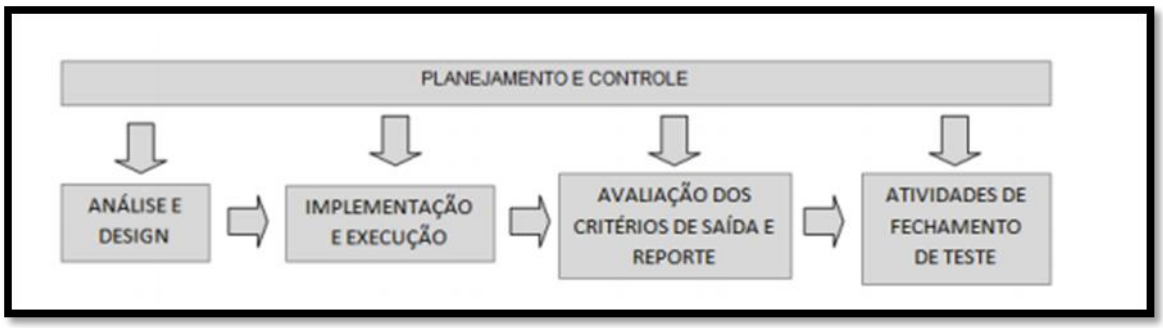
A Comissão ainda aponta que depuração é a atividade de desenvolvimento que pode identificar a causa por trás de um defeito, reparar o código e checar se os defeitos ainda existem ou se foram corrigidos corretamente. Para essa verificação é feito um teste de confirmação por um testado. As responsabilidades de cada atividade são bem distintas: testadores testam e desenvolvedores depuram. Uma

visão comum do processo de teste é de que ele consiste apenas na fase de executar o programa, na verdade, é uma parte do teste, que não contempla todas as atividades do teste. E citam como exemplos: planejamento e controle, escolha das condições de teste, modelagem dos casos de teste e checagem dos resultados, avaliação do critério de conclusão, geração de relatórios sobre o processo de teste e sobre sistema alvo de teste e encerramento ou conclusão, teste também inclui revisão de documentos (incluindo o código fonte) e análise estática. Ainda neste contexto, Graham et al. (2008, p. 20 apud SILVA; MORENO, 2011, p. 3) esclarecem que:

> O teste de software é um processo composto de uma série de atividades. Existem várias propostas de processo de teste para a abordagem tradicional na literatura, mas a maioria contém as seguintes atividades: planejamento e controle, análise e design, implementação e execução, avaliação dos critérios de saída e reporte e atividades de fechamento de teste.

Para reforçar o entendimento, observa-se a figura 1.13

Figura 1.13 - Atividades fundamentais de teste



Fonte: SILVA; MORENO, 2011, p. 3

Silva e Moreno (2011, p. 3) ainda explicam que**:**

> As atividades de planejamento dos testes englobam a determinação do escopo do teste, determinação dos riscos, dos objetivos dos testes, a abordagem do teste, definição de recursos necessários e planejamento as atividades posteriores. O controle das atividades tem como o objetivo medir e documentar o progresso das demais atividades e avaliar se o plano traçado está sendo cumprido. Na fase de análise e design, os objetivos são transformados em condições de testes. Neste momento, identifica-se o que será testado e é avaliado se o sistema e os seus requisitos são testáveis.

Segundo a Comissão Internacional para Qualificação de Teste de Software (2005), no processo de teste para a verificação da qualidade do software, diferentes

pontos de vista levam em conta diversos objetivos, por esse motivo os testes podem possuir objetivos diversos, entre eles a Comissão destaca três:

1) Encontrar defeitos.
2) Ganhar confiança sobre o nível de qualidade e prover informações.
3) Prevenir defeitos.

A Comissão Internacional para Qualificação de Teste de Software (2005, p. 13), aponta exemplos que ajudam a explicar a ideia de dois dos objetivos citados, dos quais destacam-se a procura de defeitos, mostra-se que, no teste feito em desenvolvimento (teste de componente, integração e de sistemas), o principal objetivo pode ser causar o maior número de falhas possíveis, de modo que os defeitos no software possam ser identificados e resolvidos.

Ainda, a Comissão Internacional para Qualificação de Teste de Software, 2005, p. 13) no caso da segunda hipótese, que especula sobre ganhar a confiança, é apontado que:

> No teste de aceite o objetivo principal pode ser confirmar se o sistema está funcionando conforme o esperado, ou seja, prover a confiabilidade de que esteja de acordo com o requisito. Em alguns casos o principal objetivo do teste pode ser avaliar a qualidade do software (não com a intenção de encontrar defeitos), para prover informações sobre os riscos da implantação do sistema em um determinado momento aos gestores. Testes de manutenção podem ser usados para testar se não foram inseridos erros durante o desenvolvimento de mudanças.

Após a execução dos testes, é realizada a etapa de avaliação de critérios de saída e reporte. Nela, os resultados dos testes são verificados para avaliar se a quantidade de testes foi suficiente ou se serão necessários realizar mais testes. Ainda nesta etapa é verificado se os critérios de saída estão de acordo com o especificado e um relatório provendo informação a respeito dos testes é gerado. Neste momento é verificado se o que será entregue é o que foi pedido e planejado. (SILVA; MORENO, 2011, p. 4)

O processo de teste não é realizado apenas no final do ciclo de desenvolvimento de software, mas sim em todo o ciclo. É importante ter isto em

mente, pois segundo a regra 10 de Meyers[5], o custo da correção dos defeitos tende a subir quanto mais tarde eles forem corrigidos (BASTOS et al., 2007 apud SILVA; MORENO, 2011, p. 4).

Os benefícios esperados com a implantação de um processo de teste são inúmeros destacando dentre eles a obtenção de maior qualidade dos softwares produzidos pelas empresas além de redução do tempo e capital empregado com manutenções corretivas. Informações como a quantidade de defeitos por versão de produto, quanto do software foi testado, quantidade de ciclos entre o desenvolvimento e o teste auxilia não só no melhor entendimento dos processos internos como também aumentam a confiança da equipe no produto. Além dessas contribuições, o panorama que o teste expõe traz a luz questões outrora não conhecidas e que são pertinentes para a tomada inteligente de decisões (FAPEG, 2013).

[5] **REGRA 10 DE MEYERS:** Em 1979, Glenford Myers em seu livro 'The Art of Software Testing' (A arte de teste de software), já apresentava o conceito no qual quanto mais cedo descobrimos e corrigimos o erro, menor é o seu custo para o projeto. Esse custo em correção de BUGS cresce 10 vezes para cada estágio em que o projeto do software avança.

2 METODOLOGIA

Segundo Prodanov e Freitas (2013) método científico é um conjunto de procedimentos adotados com o propósito de atingir o conhecimento.

A metodologia utilizada fundamenta-se em pesquisa bibliográfica de livros, artigos científicos e periódicos, como base teórica para início do desenvolvimento do projeto. Será desenvolvido um guia de testes de software contendo a descrição textual do sistema e um software, com objetivo de melhorar os processos de implantação e manutenção dos softwares em uma.

O projeto denominado **Guia rápido de testes de software**, tem como finalidade desenvolver um guia de testes de software para empresas de pequeno porte, que abordará a descrição dos passos necessários para realização de testes em softwares, com o intuito de garantir a qualidade e necessidades especificadas pelo cliente, o projeto terá como objetivos elaborar estudo e pesquisa sobre as características e aplicações desse tipo de ferramenta, analisar os principais benefícios que esse tipo de ferramenta de auxilio traz aos usuários, verificar as principais dificuldades e entraves para teste de software, bem como mostrar os principais aspectos que devem ser considerados para realização com sucesso de testes de software, de modo a garantir a qualidade e funcionamento adequado.

De acordo com Severino (2007) o caminho para a construção de um projeto, são etapas a serem percorridas como: a determinação do tema, levantamento de referências, leitura, construção do projeto, finalizando com a redação do texto.

A construção do projeto Guia rápido de testes de software, trata-se de uma pesquisa bibliográfica realizada para fornecer uma ferramenta de consulta para os responsáveis pelas atividades de teste, tais como desenvolvedores e analistas, além de alunos e pessoas com interesse sobre as práticas para teste de software, de maneira acessível e rápida para a realização de testes de software. Com o intuito de atingir este objetivo serão utilizados métodos e técnicas adequadas.

O desenvolvimento do projeto será baseado no uso do conhecimento adquirido ao longo do curso de Informática para Negócios e experiências vivenciadas, pois com isso será realizada a observação dos principais problemas dentro do tema proposto, e será realizado em duas partes.

A priori será realizada uma pesquisa de conteúdo teórico, baseada em um tratamento analítico-interpretativo dos dados, informações e conteúdo colhido nas pesquisas realizadas.

A seguir serão relacionados os passos percorridos na elaboração do projeto de uma forma lógica, de acordo com os conhecimentos adquiridos através das aulas, das pesquisas teóricas realizadas e no conhecimento profissional de alguns dos autores.

2.1 Apresentação do problema e justificativa

O tempo para a execução dos testes é pouco e custoso, por esse motivo, o intento para a realização deste projeto deve-se a observação da necessidade de orientação e melhor explicação sobre os principais procedimentos necessários a realização dos testes em relação ao porte do software de acordo com a empresa. Percebe-se que esta orientação, em alguns casos, demanda um grande período de tempo, não demonstra de maneira clara suas recomendações, além de não seguir uma conduta sistemática e padronizada, dificultando assim o entendimento.

O objetivo do teste de software não é somente encontrar erros, mas desenvolver atividades tais como: planejar; controlar; analisar; implementar; executar; avaliar critérios de saída e reportar resultados; avaliar conclusão de testes; gerar relatórios e revisão de documentos.

2.2 Etapas para desenvolvimento do projeto

Após a definição do tema pelos integrantes do grupo, foram elaboradas as etapas para o desenvolvimento e construção sistemática do projeto, como segue:

1) Levantamento de referências: consultas em bibliotecas para o levantamento de livros com fontes confiáveis, baseada em autores reconhecidos, artigos acadêmicos e outras monografias, além de buscas em sites especializados e consulta à profissionais das áreas relacionadas ao tema.

2) Pesquisa: Após a escolha das bibliografias são escolhidos os assuntos que dão a base a para elaboração da fundamentação teórica do projeto.

3) Orientação: início da orientação técnica fornecida pelo professor orientador para a construção do projeto.

4) Construção: Com base nos conhecimentos adquiridos, nas pesquisas realizadas, bibliografias consultadas será executado o desenvolvimento do projeto.

5) Finalização: redação final do projeto, elaboração de considerações com o intuito de evidenciar a necessidade e importância do projeto

2.3 Cronograma do projeto

Este projeto seguirá o cronograma conforme descrito no quadro 2.1.

Quadro 2. 1 – Cronograma do projeto

DATA	ATIVIDADE
23/02/2015	Montagem dos grupos e definição do tema do projeto
09/03/2015	Entrega da 1ª versão do TCC para correção e orientação (Pré-textuais)
30/03/2015	Entrega da 2ª versão do TCC para correção e orientação (Pré-textuais + Introdução + Fundamentação teórica)
27/04/2015	Entrega da 3ª versão do TCC (Pré-textuais + Introdução + Fundamentação teórica + Metodologia)
18/05/2015	Qualificação para Defesa Oral
08/06/2015	Pré-banca - Defesa oral do projeto Apresentação do projeto para a banca examinadora
2º. Semestre de 2015	Desenvolvimento e defesa do projeto

Fonte: Autoria Própria, 2015.

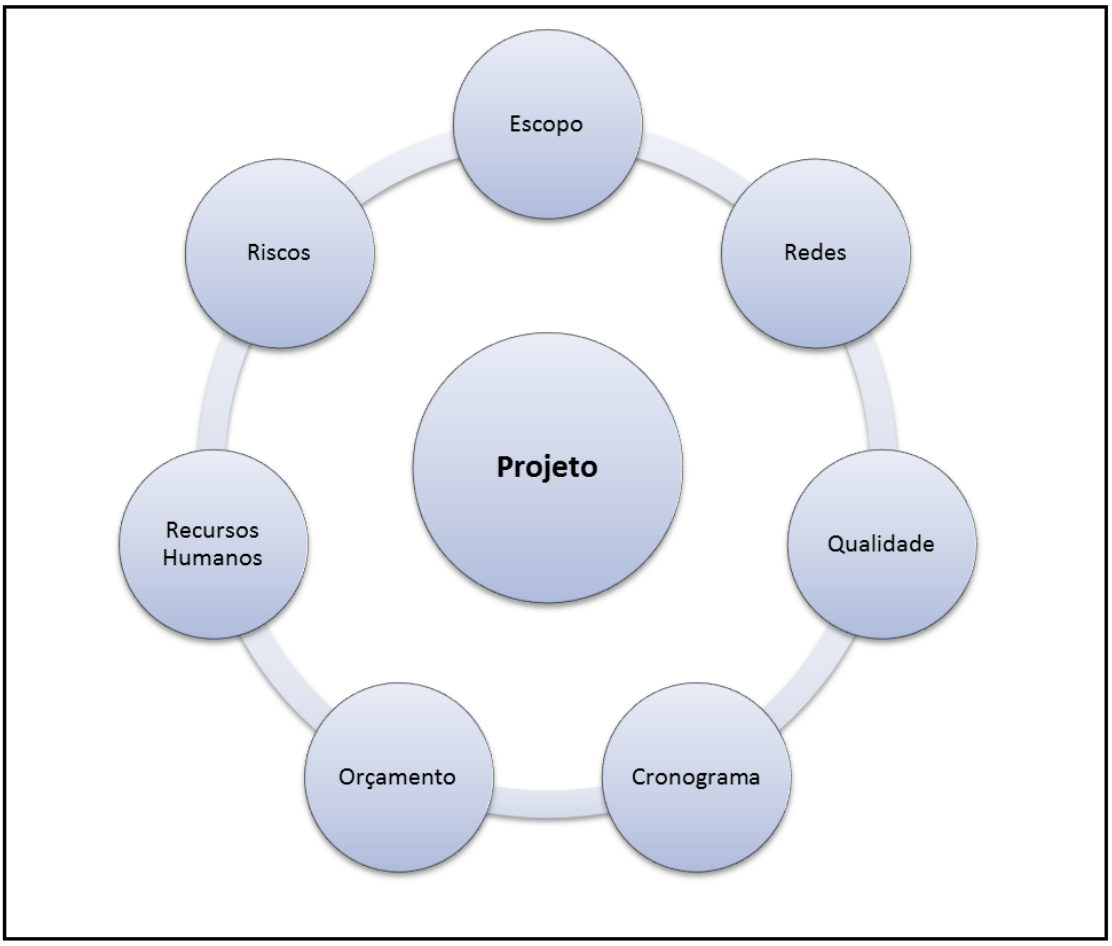
3 DESENVOLVIMENTO

De acordo com os estudos realizados, foi identificado que a elaboração de um guia rápido de testes de software, sugere o uso de uma ferramenta de apoio para as empresas que pretendem utilizar com eficiência todos os recursos de um software. Segundo Rios e Moreira (2013, p. 10),

> Na prática se pode testar um software por completo e garantir que ele ficará livre de bugs. É quase impossível testar todas as possibilidades de formas e alternativas de entradas de dados, bem como testar as diversas possibilidades e condições estabelecidas pela lógica do software.

As atividades de teste possuem um papel fundamental no sucesso do projeto de desenvolvimento de software, a figura 3.1, demonstra de maneira simples, clara e igualmente objetiva o impacto do teste no projeto de desenvolvimento como um todo.

Figura 3. 1- Estrutura de um projeto



Fonte: Autoria própria, 2015

1) A Infraestrutura de Redes influencia no funcionamento e o rendimento de um software, portanto várias situações e cenários devem ser abalizados. Esta relacionado com os Hardware que se comunicam com o Software.

2) O teste de software estará diretamente ligado à qualidade final do produto.

3) A estruturação dos testes da maneira inadequada pode gerar um alongamento não planejado no prazo de desenvolvimento e consequentemente encarecer o custo do projeto com manutenções corretivas.

4) A equipe envolvida nos testes deve ser qualificada e ciente da sua importância no processo como um todo.

5) Seguindo esse raciocínio a implementação inadequada ou errada dos testes é um risco que sempre deve ser considerado. Ou seja, o escopo do projeto no todo tem como um de seus mais relevantes aspectos o teste bem planejado e executado.

Sendo assim, Sampaio e Mancini (2007), explicam que o processo de desenvolvimento de estudo de revisão inclui caracterizar cada estudo selecionado, avaliar a qualidade destes estudos, identificar conceitos importantes, concluir sobre o que a literatura informa entre outros objetivos.

A escolha do uso de um guia rápido de testes de software foi levada por considerações como constantes problemas identificados durante a implantação e utilização de software, além do fator segurança ser necessário para um projeto sólido e não suscetível a ameaças de alteração de dados e informações. Sendo assim, será descrito as etapas de construção do projeto proposto, que trata-se da elaboração de um guia de rápido acesso e fácil utilização para teste de software, que está disponível no apêndice A, visando facilitar e padronizar a atividade de testes em empresas de pequeno porte.

3.1 Primeira etapa: Pesquisa das Ferramenta de teste

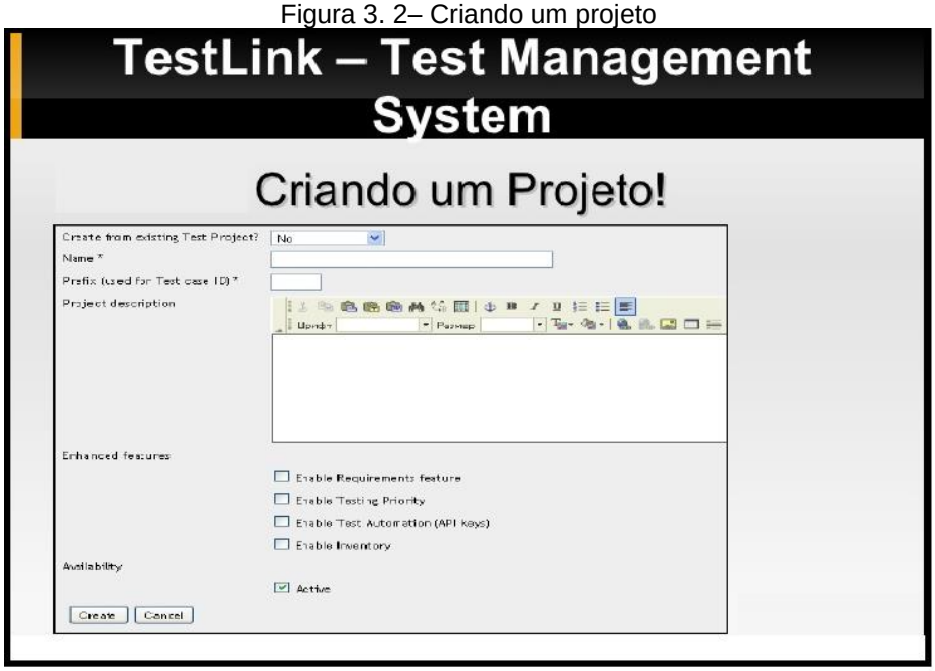
O teste de software tem por objetivo revelar e identificar falhas do produto, afim de corrigi-los. Foram pesquisadas algumas ferramentas que possuem o intuito de auxiliar tarefas praticadas no dia a dia do teste de software, como: a abertura de defeitos, execução de testes, planejamento de casos de testes, entre outros. Dentre as ferramentas pesquisadas, destacam-se: Mantis, utilizada para gestão de defeitos, TestComplete, para automatização de testes e por fim, o TestLink, utilizado para a gestão de testes. Neste projeto será utilizada a ferramenta de gerenciamento de testes TestLink, por se tratar de uma ferramenta open source, ou seja uma ferramente livre e gratuíta, e de fácil utilização, como será descrito posteriormente. Além destes fatores, a ferramenta TestLink, permite que equipes trabalhem de forma sincronizada, mesmo em locais distintos, possui controle de execução, garantindo uma base histórica dos testes aos quais a aplicação foi submetida.

3.2 Segunda etapa: TestLink

A ferramenta para testes TestLink foi a escolhida por vários fatores já citados, e para melhor interpretar a escolha serão apresentadas algumas telas, referentes ao processo para efetivação de testes, apresentando a facilidade de utilização desta ferramenta.

A figura 3.1 apresenta o layout da tela de criação de um projeto de teste. Nela é possível especificar e detalhar o seu projeto.

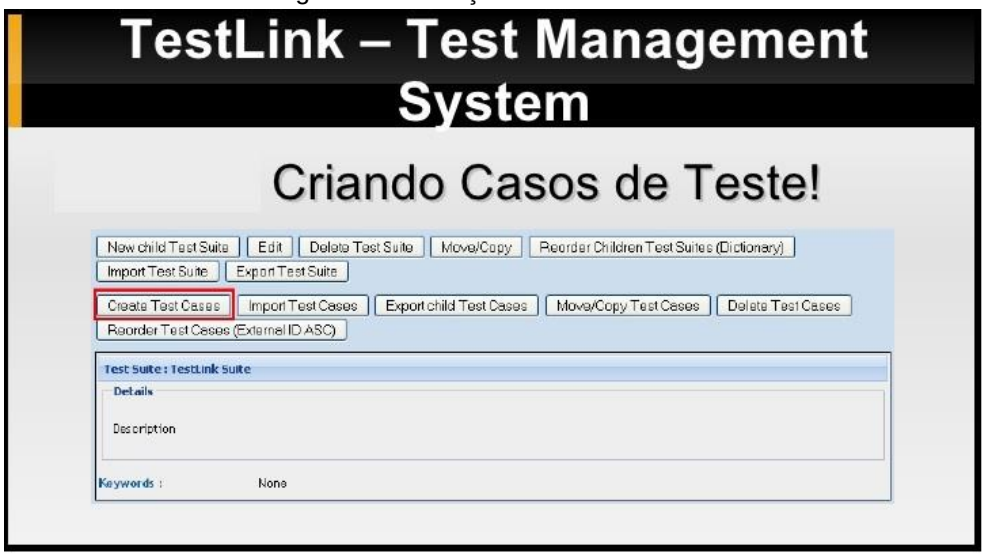
Figura 3. 2– Criando um projeto



Fonte: http://pt.slideshare.net, 2015.

A figura 3.2 apresenta o layout da tela do TestLink que realiza a criação de um caso de teste. Nesse momento é possível especificar as diretrizes do teste, definindo seus parâmetros e as especificações do caso de teste a ser desenvolvido.

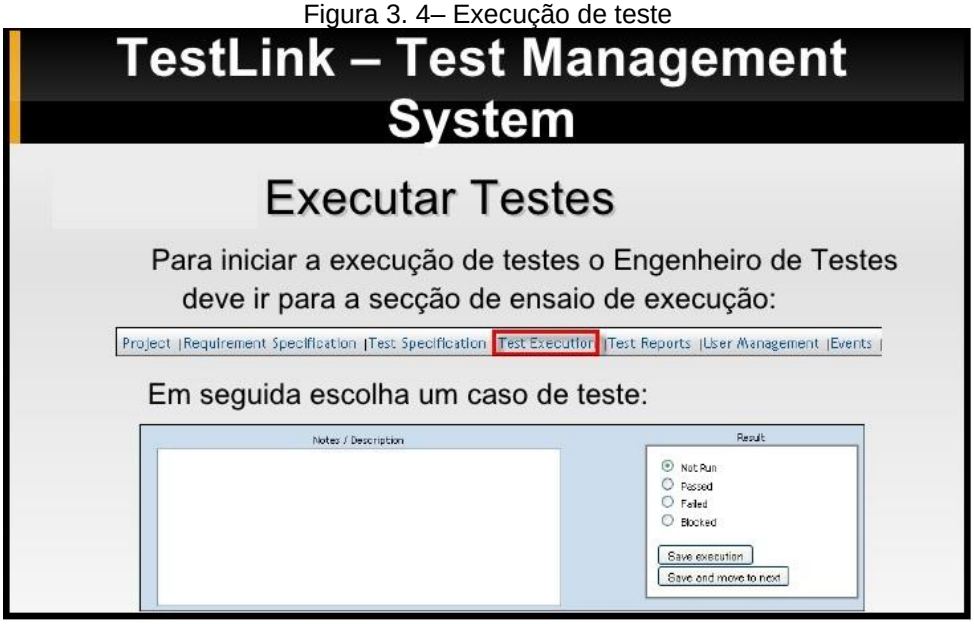
Figura 3. 3- Criação de Caso de teste



Fonte: http://pt.slideshare.net, 2015.

Na figura 3.3, inicia-se a execução do teste criado no projeto.

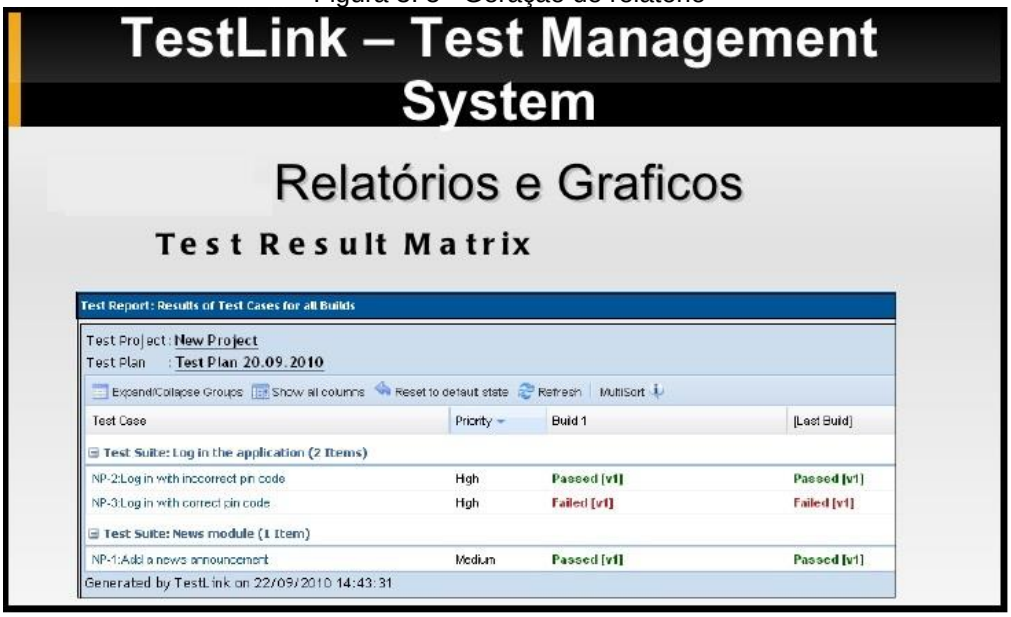
Figura 3. 4– Execução de teste



Fonte: http://pt.slideshare.net, 2015.

Na figura 3.4, é exibida a tela que permite a geração dos relatórios e gráficos que podem ser emitidos pelo TestLink, de modo a facilitar o entendimento dos testes realizados e ressaltas os principais aspectos inerentes aos testes.

Figura 3. 5– Geração de relatório



Fonte: http://pt.slideshare.net, 2015.

Conforme foi ilustrado, a ferramenta de teste TestLink é de muito fácil utilização e visualização, por este motivo, entre outros, foi escolhida entre tantas opções que existem, para ser apresentada no guia rápido de teste de software.

3.3 Terceira Etapa: Tipos de testes

Para efeito de simplificação do guia serão abordados apenas algumas categorias e alguns tipos de testes. Conforme quadro 3.1, existem muitos outros tipos de testes que são possíveis de serem efetuados, entretanto, é preciso entender os requisitos funcionais e não funcionais (garantia e utilidade) do negócio, para definir exatamente o nível de testes que se pretende estabelecer para uma determinada aplicação.

Quadro 3.1**–** Tipos de testes

Tipo de Teste	Descrição
Teste de Unidade	Teste em um nível de componente ou classe. É o teste cujo objetivo é um “pedaço do código”.
Teste de Integração	Garante que um ou mais componentes combinados (ou unidades) funcionam.
Teste Operacional	Garante que a aplicação pode rodar muito tempo sem falhar.
Teste de regressão	Toda vez que algo for mudado, deve ser testada toda a aplicação novamente.
Teste de caixa preta	Testar todas as entradas e saídas desejadas. Não se está preocupado com o código, cada saída indesejada é vista como um erro.
Teste caixa branca	O objetivo é testar o código. Às vezes, existem partes do código que nunca foram testadas.
Teste Funcional	Testar as funcionalidades, requerimentos, regras de negócio presentes na documentação.

Teste de Interface	Verifica se a navegabilidade e os objetivos da tela funcionam como especificados.
Teste de Performance	Verifica se o tempo de resposta é o desejado para o momento de utilização da aplicação.
Teste de carga	Verifica o funcionamento da aplicação com a utilização de uma quantidade grande de usuários simultâneos.
Testes de stress	Avalia o comportamento do software em tempo de situação crítica (gargalos).
Testes de Configuração	Testar se a aplicação funciona corretamente em diferentes ambientes de hardware ou de software.
Testes de Segurança	Testar a segurança da aplicação das mais diversas formas.

Fonte: adaptado de BARBOSA e TORRES, 2011, p. 9

3.4 Quarta etapa: Análise de Fluxo

Um modelo de processo de desenvolvimento de software pode ser visto como uma representação das atividades envolvidas na criação de um software, além disso, é uma forma fácil de representar o gerenciamento e progresso do projeto. Para isto existem alguns tipos de metodologias e modelos criados para a efetiva construção de um software, entre as quais se destacam:

1) Metodologia Ágil – O desenvolvimento ágil providencia uma estrutura de desenvolvimento rápida chamados de interação, que é como que cada interação fosse um projeto, e que possui todas as tarefas necessárias para implantar o incremento da nova funcionalidade;

2) Metodologia Scrum [6]– Esta metodologia é utilizada pela empresa, quando fica difícil o planejamento à frente, mecanismos de feedback constituem o centro da

[6] **Metodologia Scrum:** É uma metodologia que possui seu foco no gerenciamento e projeto da organização onde é difícil planejar à frente.

técnica de gerenciamento. O Scrum não descreve o que fazer em cada situação, ele é usado para trabalhos complexos nos quais é impossível prever tudo que irá ocorrer;

3) Método Cascata - É um método de desenvolvimento sequencial que flui constantemente (como uma cascata) através de fases de análise de requisitos, projeto, implementação, testes, integração e manutenção do software;

4) Modelo Espiral – É uma evolução do modelo cascata, onde se argumentava ser um convite para falhas, esta evolução é usada com maior frequência em grandes projetos uma vez que ele tem uma série de vantagens, apesar da sua simplicidade, ele suporta mecanismos de redução de riscos, inclui interações, apresenta abordagens sistemáticas, é mais versátil para lidar com mudanças, e os engenheiros de software podem começar o trabalho no sistema, mais cedo.

Existem outras metodologias e modelos de desenvolvimento, entretanto, o intuito do guia desenvolvido é de abordar de uma maneira geral as principais técnicas para a realização de testes de software, bem como suas aplicações sem se prender a um modelo ou metodologia especifica de desenvolvimento.

3.5 Quinta etapa: Demonstração do Guia

Inicialmente foram criados layouts de telas do sistema, o modelo que foi previamente desenhado. A criação do guia como primeiro passo, permitiu uma melhor visualização do que deveria ser programado a seguir. Esse primeiro passo de Design é importante para que seja claro os objetivos do desenvolvimento do Sistema.

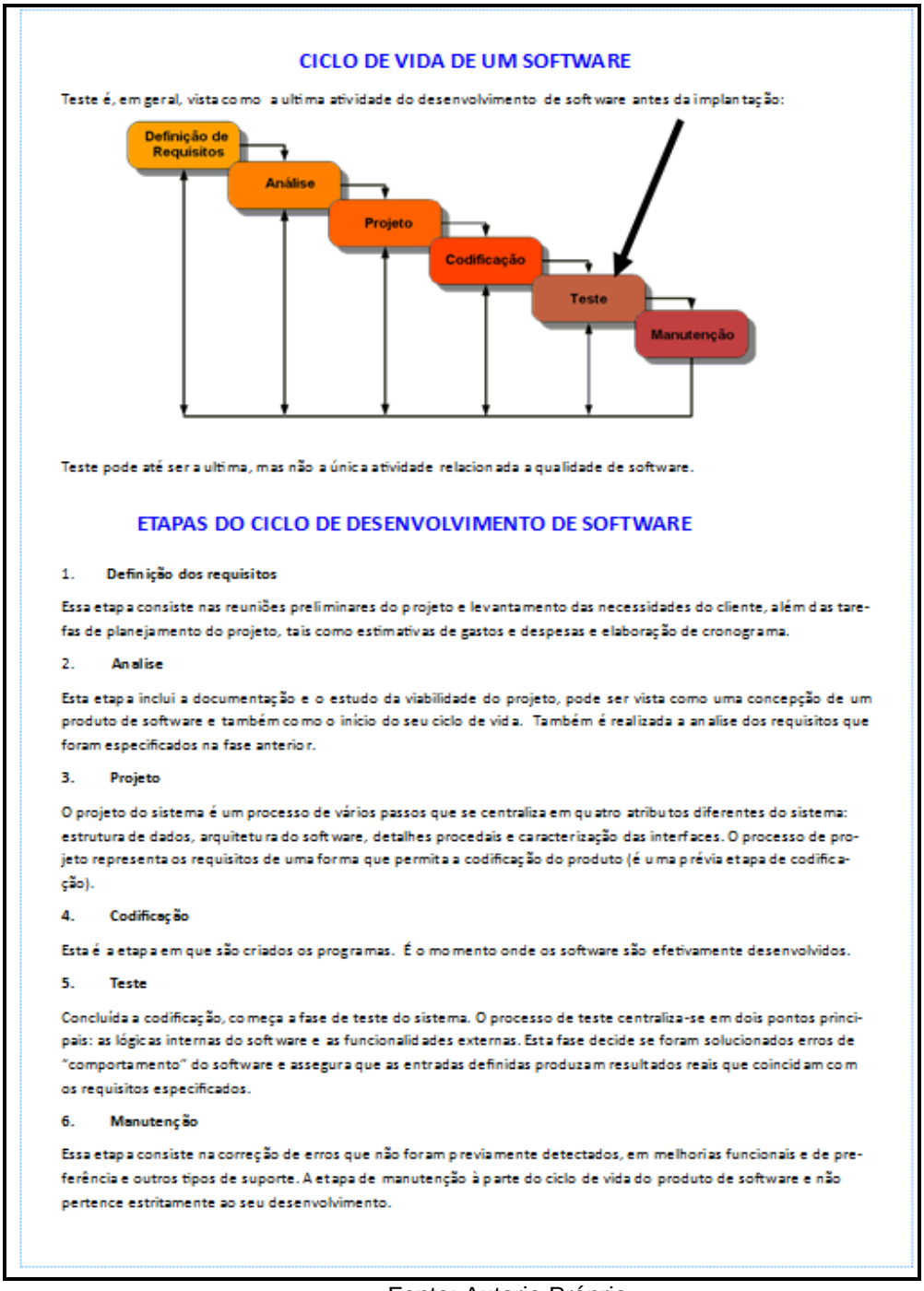
A figura 3.5 apresenta a capa do guia, referenciando a FATEC, instituição que proporcionou recursos para desenvolver esse guia.

Figura 3. 6 – Capa do Guia

Fonte: Autoria Própria

A figura 3.6 apresenta de maneira interativa e objetiva o ciclo de vida de um software, apresentando as características das etapas e salientando a estruturação do desenvolvimento de um software de modo a revelar o impacto que testes eficazes e eficientes agregarão ao software.

Figura 3. 7 – Página 6: Ciclo de Vida de um software



CICLO DE VIDA DE UM SOFTWARE

Teste é, em geral, vista como a ultima atividade do desenvolvimento de software antes da implantação:

Teste pode até ser a ultima, mas não a única atividade relacionada a qualidade de software.

ETAPAS DO CICLO DE DESENVOLVIMENTO DE SOFTWARE

1. **Definição dos requisitos**

Essa etapa consiste nas reuniões preliminares do projeto e levantamento das necessidades do cliente, além das tarefas de planejamento do projeto, tais como estimativas de gastos e despesas e elaboração de cronograma.

2. **Analise**

Esta etapa inclui a documentação e o estudo da viabilidade do projeto, pode ser vista como uma concepção de um produto de software e também como o início do seu ciclo de vida. Também é realizada a analise dos requisitos que foram especificados na fase anterior.

3. **Projeto**

O projeto do sistema é um processo de vários passos que se centraliza em quatro atributos diferentes do sistema: estrutura de dados, arquitetura do software, detalhes procedais e caracterização das interfaces. O processo de projeto representa os requisitos de uma forma que permita a codificação do produto (é uma prévia etapa de codificação).

4. **Codificação**

Esta é a etapa em que são criados os programas. É o momento onde os software são efetivamente desenvolvidos.

5. **Teste**

Concluída a codificação, começa a fase de teste do sistema. O processo de teste centraliza-se em dois pontos principais: as lógicas internas do software e as funcionalidades externas. Esta fase decide se foram solucionados erros de "comportamento" do software e assegura que as entradas definidas produzam resultados reais que coincidam com os requisitos especificados.

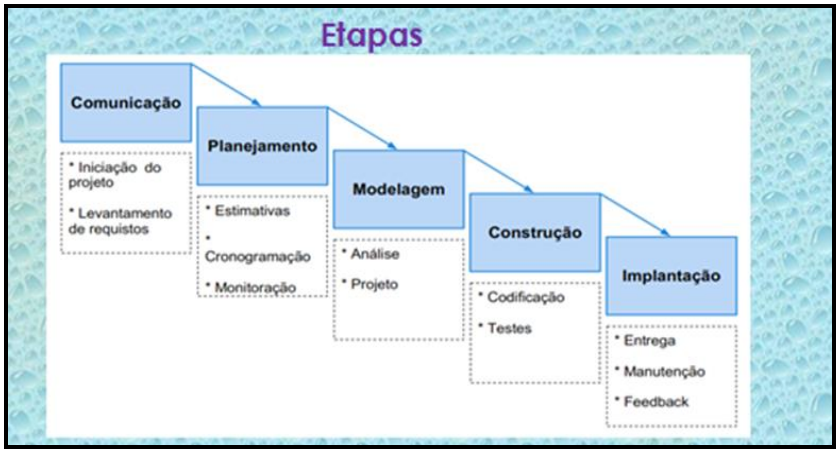
6. **Manutenção**

Essa etapa consiste na correção de erros que não foram previamente detectados, em melhorias funcionais e de preferência e outros tipos de suporte. A etapa de manutenção à parte do ciclo de vida do produto de software e não pertence estritamente ao seu desenvolvimento.

Fonte: Autoria Própria

A figura 3.7 descreve como é o ciclo clássico de desenvolvimento de software, o qual é uma técnica que utiliza uma abordagem sequencial e sistemática para desenvolvimento de software.

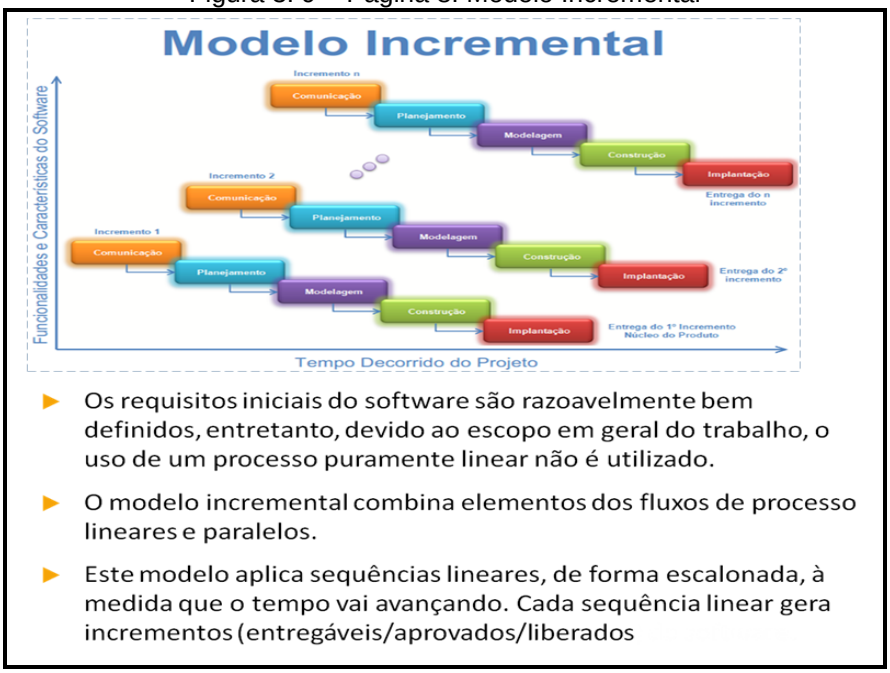
Figura 3. 8 – Página 7: Modelo Cascata



Fonte: Autoria Própria

Outro conteúdo que deve ser mencionado está na figura 3.8 que apresenta uma outra metodologia muito utilizada para desenvolvimento de software:

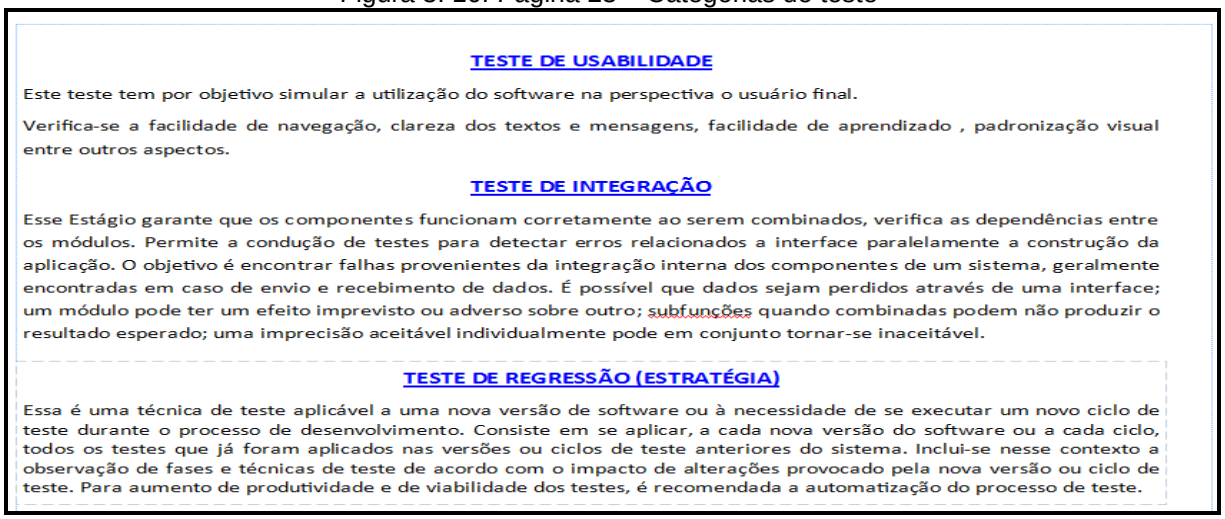
Figura 3. 9 – Página 8: Modelo Incremental



Fonte: Autoria Própria

Finalizando essa seção a figura 3.9, apresenta um fragmento da página 10 onde são conceituados os principais tipos de teste, buscando melhorar o entendimento do leitor e o orientar sobre a aplicabilidade de cada categoria de teste.

Figura 3. 10: Página 23 – Categorias de teste



TESTE DE USABILIDADE

Este teste tem por objetivo simular a utilização do software na perspectiva o usuário final.

Verifica-se a facilidade de navegação, clareza dos textos e mensagens, facilidade de aprendizado , padronização visual entre outros aspectos.

TESTE DE INTEGRAÇÃO

Esse Estágio garante que os componentes funcionam corretamente ao serem combinados, verifica as dependências entre os módulos. Permite a condução de testes para detectar erros relacionados a interface paralelamente a construção da aplicação. O objetivo é encontrar falhas provenientes da integração interna dos componentes de um sistema, geralmente encontradas em caso de envio e recebimento de dados. É possível que dados sejam perdidos através de uma interface; um módulo pode ter um efeito imprevisto ou adverso sobre outro; subfunções quando combinadas podem não produzir o resultado esperado; uma imprecisão aceitável individualmente pode em conjunto tornar-se inaceitável.

TESTE DE REGRESSÃO (ESTRATÉGIA)

Essa é uma técnica de teste aplicável a uma nova versão de software ou à necessidade de se executar um novo ciclo de teste durante o processo de desenvolvimento. Consiste em se aplicar, a cada nova versão do software ou a cada ciclo, todos os testes que já foram aplicados nas versões ou ciclos de teste anteriores do sistema. Inclui-se nesse contexto a observação de fases e técnicas de teste de acordo com o impacto de alterações provocado pela nova versão ou ciclo de teste. Para aumento de produtividade e de viabilidade dos testes, é recomendada a automatização do processo de teste.

Fonte: Autoria Própria

3.6 Resultados obtidos

Após uso do guia foi constatado uma melhora significativa tanto na qualidade do processo, quanto no tempo para finalização do software, diminuindo a curva de aprendizado, o tempo desperdiçado para tirar dúvidas, a necessidade de re-testes e consequentemente todo o custo relativo ao processo de construção de um software.

Foi identificado que o guia poderá dar suporte para identificar caminhos que representam o comportamento excepcional de um software, assim como a detecção de erros que são aplicados aos caminhos, os quais representam o comportamento normal e com isso agregar um aumento na qualidade e eficiência no uso do software.

Realizar testes não consiste simplesmente na geração e execução de casos de testes, mas envolvem também questões de planejamento, gerenciamento e análise de resultados. Após ter uma estratégia de testes definida é muito importante buscar ferramentas que se encaixam a esta estratégia. Além disto, cada software

possui características específicas que devem ser consideradas no momento da realização dos testes.

Os testes de software poderão ser realizados durante todo o ciclo de desenvolvimento e implantação, utilizando diversos níveis e tipos de rigor e detalhe. Podemos testar desde a estrutura interna até a funcionalidade básica, passando pela interface com o usuário.

Para melhor adequação a estrutura e porte da empresa devem-se escolher as técnicas e categorias de teste que melhor se adaptem ao modelo do sistema e ciclo de vida do mesmo, afim de adequar as constantes evoluções e alterações que venha acontecer.

No mercado competitivo e exigente, com uma demanda cada vez maior por sistemas e softwares cada vez mais complexos, com prazos limitados e custos elevados, a atividade de teste de software tornou-se vital para o ciclo de desenvolvimento e sucesso na implantação e utilização.

CONSIDERAÇÕES FINAIS

O processo de desenvolvimento de software está intimamente ligado a filosofia da empresa na qual está implantado, pontos como linguagens de programação, cultura, ferramentas, paradigmas, arquiteturas, metodologias exercem influência, em menor ou em maior grau, sobre o processo de desenvolvimento de software. Embora haja frameworks específicos e normas internacionais que regem a criação de processos, em cada empresa ele será dotado de peculiaridades e modificações que devem estar de acordo com a necessidade do negócio atendido por essas empresas. Logo o processo de teste também é dinâmico, sendo influenciado por diversos fatores internos. Isso ocorre, pois, o teste possui ligação com o processo de desenvolvimento, e uma vez que o primeiro é fundamentalmente customizado o segundo também acompanha essa necessidade de ajustes próprios.

Após ter sido observado esses pontos, alguns desafios foram encontrados quanto ao desenvolvimento do projeto assim como em sua implantação nas empresas. O primeiro dos desafios se encontra no campo do nível da similaridade entre as empresas, mesmo sendo o foco deste guia ser as microempresas cada uma dessas possui uma finalidade de negócio, cultura e tecnologias distintas. Essas diferenças então dificultam a proposição de um processo único, que sirva sem modificações, para todas elas.

Dado a heterogeneidade existente entre as empresas que, a partir da proposição do projeto, cada uma realize os ajustes para que o processo se encaixe melhor em suas realidades. É importante entender que o teste de software não tem por objetivo primordial, como é de senso comum, mostrar que o produto está correto. Ao contrário do senso comum, o objetivo do teste é fazer o produto falhar, ou seja, descobrir, demonstrar e documentar os defeitos desse produto, neste âmbito o guia rápido de teste de software veio trazer de uma forma simples e sistemática uma ferramenta para conseguir sanar dúvidas em pouco tempo.

Outro ponto que merece observação quanto à implantação dos testes é a impressão equivocada de que com a implantação dos testes todos os problemas e deficiências das empresas relacionados a produção de software serão resolvidos, isso significa que não existe uma forma, uma metodologia, tecnologia ou regra mágica que seja adequada para todas as realidades e que resolva todos os problemas existentes no processo de software. O teste de software pode ser enxergado como uma ferramenta em prol da qualidade o qual tem o papel de validar e verificar o que é produzido, influenciando positivamente na qualidade dos produtos. Porém, o teste sozinho não é capaz de solucionar todas as falhas e os problemas de qualidade, uma vez que essa está intimamente ligada a vários outros fatores. Assim, como qualquer outra ferramenta o teste deve ser utilizado em conjunto com outras políticas e técnicas de forma correta e efetiva, para que sejam alcançados os resultados de melhoria esperados e assim criar e aplicar um processo de teste de software, adequado a empresa.

Foi constatado após a implantação em uma empresa onde foram realizadas as observações, que os testes sendo executados sistematicamente com o apoio do guia têm fornecido dados e informações relevantes à melhoria tanto do próprio processo de teste quanto no de desenvolvimento do projeto, aumentando sua eficiência e abrangência, permitindo capacitar pessoal para desenvolver as atividades de teste com mais segurança e autoconfiança. Pelo lado de investimento o processo de teste tende a se pagar através do contínuo aperfeiçoamento e incorporação de novas ferramentas de acordo com a empresa.

Após análise deste projeto, constatou-se a efetiva vantagem da utilização de um guia de rápida visualização das tarefas e para a continuidade deste estudo sugere-se que seja elaborado um manual que defina mais profundamente as categorias e tipos de testes e crie novos planos de testes especialmente para os profissionais com menos experiência na área, visto que a economia em termos de tempo e custos foi bem relevante. Abrindo espaço para a elaboração de projetos específicos e mais abrangentes, para o mundo coorporativo.

REFERÊNCIAS

ALBAGLI, S.; LASTRES, H. M. M. **Informação e globalização na era do conhecimento**. Rio de Janeiro: Campus, 1999.

ALECRIM, E. **O que é tecnologia da informação**, 2011. Disponível em <http://www.infowester.com/ti.php>. Acesso em 20 mar. 2015.

BARBOSA, Filipe B.; TORRES, Isabelle V. **O Teste de software no mercado de trabalho**, 2011. Disponível em <http://www.revista.faculdadeprojeção.edu.br. >. Acessado em 19 out. 2015.

BARTIÉ, A. **Garantia de qualidade de software**: adquirindo maturidade organizacional. Rio de janeiro: Elsevier, 2002.

BASTOS, A. et al. **Base de conhecimento em teste e software,** 3 ed. São Paulo: Martins, 2012

BRIGNOL, S.M.S. **Novas tecnologias de informação e comunicação nas relações de aprendizagem da estatística no ensino médio.** 2004. 152 p. Monografia Especialização – Faculdade Jorge Amado, Salvador, 2004.

BRITO, G. S.; KNOLL, A. C. G. **Afinal, professor o que é tecnologia.** Disponível em <http://www.gazetadopovo.com.br/blogs/educacao-e-midia> Acesso em: 20 mar. 2015.

COLOMBO, R. M. T; GUERRA, A. C. **Tecnologia da informação:** qualidade de produto de software. Brasília: PBQP, 2009.

COMISSÃO INTERNACIONAL PARA QUALIFICAÇÃO DE TESTE DE SOFTWARE. **Base de Conhecimento para Certificação em Teste**, 2005. Disponível em: <https://www.passeidireto.com/>. Acesso em: 30 jul. 2015.

CHIAVENATO, I. **Teoria Geral da Administração**. Rio de Janeiro: Campus, 2003.

CROWDTEST. Disponível em <http://crowdtest.me/> Acesso em: 27 ago. 2015.

FAPEG. **Manual do Processo de Teste de Software para Micro e Pequena Empresas versão 2.0**. Goiânia-Goiás, Brasil, 2013.

FILOCZAR. Disponível em <http://www.filoczar.com.br/Apostilas/Periféricos.pdf> Acesso em: 25 abr. 2015

FOINA, P.R. **Tecnologia de informação:** planejamento e gestão - São Paulo: Atlas, 2001.

GOOGLE. Disponível em <www.google.com/>. Acesso em: 17 abr. 2015.

HOLANDA, A. B. **Minidicionário Aurélio da língua portuguesa.** 4. ed.Rio de Janeiro: Nova Fronteira, 2001

HOUAISS, A.; VILLAR, M. S. **Minidicionário Houaiss da língua portuguesa.** 4. ed. Rio de Janeiro: Objetiva, 2010

ICARRO. Disponível em <http://www.icarros.com.br/fiat/uno/2012> Acesso em: 17 abr. 2015.

IME, 1996. Disponível em <http://www.ime.uerj.br/~demoura/Especializ/Ariane/> Tradução: José Nivaldo Hinkel. Acesso em 17 abr.2015.

INTEL. Disponível em <http://www.itel.com>, acesso em 10 mai. 2015.

JAVA. Disponível em <http://www.java.com>, acesso em 10 mai. 2015.

KARASINSKI, l. **O que é tecnologia**. Tecmundo, 2013. Disponível em: <http://www.tecmundo.com.br/tecnologia/42523-o-que-e-tecnologia-.html. >. Acesso em: 20 mar.2015.

KENN, P. G. W. **Guia Gerencial para a tecnologia da informação: Conceitos essenciais e terminologia para empresas e gerentes**. Rio de Janeiro: Campus, 1996.

KOSCIANSKI, A.; SOARES, M.S. **Qualidade de Software:** aprenda as metodologias e técnicas mais modernas para o desenvolvimento de Software. 2 ed. São Paulo: Novatec, 2007.

LAUDON, J.P.; LAUDON, K.C. **Sistemas de informação gerencial:** Administrando a empresa digital. 5 ed. São Paulo: Pearson Education do Brasil, 2004.

LEITE, A. **Metodologia do Desenvolvimento de Software,** 2013. Disponível em <http://www.devmedia.com.br/metodologia-de-desenvolvimento-de-software/1903> Acesso em: 11 mai. 2015

MAÑAS, A.V. **Administração de sistemas de informação:** Como otimizar a empresa por meio dos sistemas de informação. 8 ed. São Paulo: Érica, 2011.

MANCINI, M. C. ; SAMPAIO, R. F.**Guia para estudos de revisão sistemática**: uma opção metodológica para as Ciências do Movimento Humano. Disponível em <http://www.seer.ufrgs.br/Movimento/article/viewFile/41542/28358> Acesso em: 22 abr.2015.

MENDOÇA, A.; ZELENOVSKY, R. **PC: um Guia Prático de:** Hardware e Interfaceamento. 4. ed. Rio de Janeiro: MZ, 2006.

MILES, R.; PILONE, D. **Use a Cabeça:** Desenvolvimento de Software. Rio de Janeiro: Alta Books, 2008.

NOÇÕES DE ENGENHARIA DE SOFTWARE. Disponível em: <http://nocoesengesw.blogspot.com.br/>. Acesso em 18 abr. 2015.

OFICINA DA NET. Disponível em:<http://www.oficinadanet.com.br/artigo/1908/>. Acesso em: 20 abr. 2015.

PAULINO, D. **Tipos de software: Você realmente sabe o que é um software?** Disponível em:<http://www.oficinadanet.com.br/artigo/1908/>. Acesso em: 20 abr. 2015.

POTTER, R.E.; RAINER, R.K.; TURBAN, E. **Administração de tecnologia da informação:** teoria e pratica. 3 ed. Rio de Janeiro: Elsevier, 2005.

PRESSMAN, R.S. **Engenharia de Software:** Uma Abordagem Profissional. 7 ed. Porto Alegre: AMGH, 2011.

PRIBERAM. Disponível em <http://www.priberam.pt/DLPO/falha>. Acesso em: 30 jul. 2015.

PRIBERAM. Disponível em <http://www.priberam.pt/DLPO/defeito>. Acesso em: 30 jul. 2015.

PRODANOV, C. C.; FREITAS, E. C. **Metodologia do Trabalho Científico**: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico. 2ª Edição. Rio Grande do Sul: Universidade Feevale, 2013.

REYNOLDS, G. W.; STAIR, R. M. **Princípios de sistemas de informação**. 9 ed. São Paulo: Cengage Learning, 2011.

RIOS, E.; MOREIRA T. **Teste de software.** 3 eds. Revisada e completada. Rio de Janeiro: Alta Books, 2013.

SANTOS NETO, P. A. **Introdução à Engenharia de Software,** 2007. Disponível em <http://www.ufpi.br/subsiteFiles/pasn/arquivos/files/IntroducaoEngenhariaDeSoftware.pdf> Acesso em: 22 abr. 2015.

SEGUNDO, P. Disponível em <http://pt.slideshare.net/placydtwo/test-link-12593105>. Acesso em: 13 set. 2015.

SEVERINO, A. J. **Metodologia do trabalho científico.** 23 ed. São Paulo: Cortez, 2007.

SIGNIFICADOS. Disponível em <http://www.significados.com.br/hardware/>. Acesso em: 17 abr. 2015.

SILVA, M.F.; MORENO, A.G. **Automação em testes ágeis**. Disponível em: <www.revistas.unifacs.br/index.php/rsc/article/download/1903/1493>. Acesso em: 30 jul. 2015.

UNIVERSIDADE FEDERAL DO PARÁ. Disponível em <http://www.ufpa.br/dicas/mic/mic-proc.html> Acesso em: 25 abr. 2015.

TONSIG, S. L. **Engenharia de Software**: Análise e projeto de sistemas. São Paulo: Futura, 2003.

APÊNDICES

APÊNDICE A – GUIA RÁPIDO DE TESTES DE SOFTWARE

SUMÁRIO

INTRODUÇÃO

Este guia pretende apresentar e orientar o melhor caminho a ser seguido na elaboração e execução dos testes, visando eficiência, exatidão, segurança, rapidez e redução de custos adicionais derivados de defeitos e falhas. O guia pretende ainda estabelecer a ligação dos testes de softwares à qualidade do produto final, abrangendo vários tipos de testes, as técnicas utilizadas em sua execução e os benefícios para as empresas que adotam os testes no seu cotidiano. Após verificar que em algumas empresas, principalmente as micros e pequenas, há uma lacuna referente à quais testes devem ser aplicados, foi identificada uma grande vantagem em utilizar um guia de rápida visualização das tarefas, propondo um caminho padrão a ser seguido na execução de teste, cuja a importância é fundamental para a qualidade do software, diminuindo os erros e defeitos que contribuam para o aumento de sua credibilidade. O material a ser apresentado é destinado aos profissionais, professores, estudantes e demais pessoas interessadas em aprender mais sobre os testes de software e sua relação com a confiabilidade e eficiência do software desenvolvido.

Das diversas fases associadas à Engenharia de Software, o Teste é a fase indispensável para validar e verificar sistemas em desenvolvimento. Desde o estabelecimento desta noção, tem-se feito grande volume de pesquisa em torno deste assunto. Uma das questões mais importantes na execução de um teste é caracterizar quais (e quantos) testes são suficientes para garantir qualidade do produto.

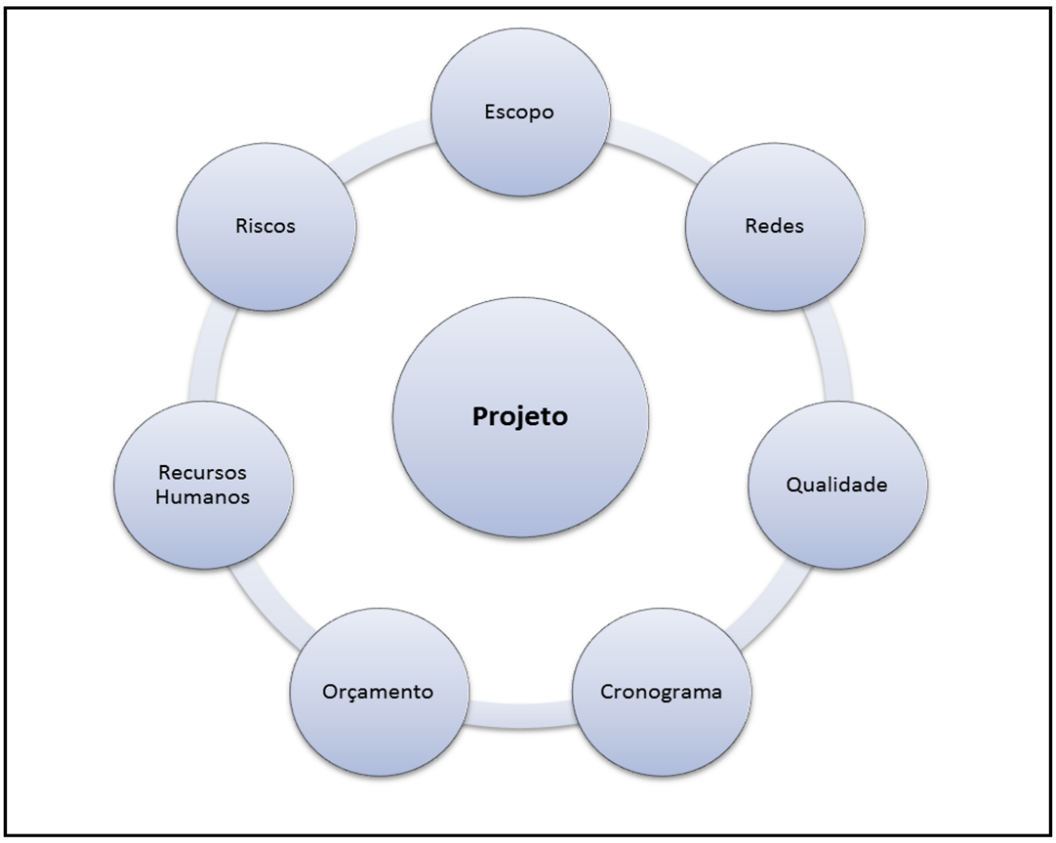
No processo de teste para a verificação da qualidade do software, diferentes pontos de vista levam em conta diversos objetivos, por esse motivo os testes podem possuir objetivos diversos, entre eles pode-se citar três:

1. Encontrar defeitos.
2. Ganhar confiança sobre o nível de qualidade e prover informações.
3. Prevenir defeitos.

Um exemplo que ajuda a explicar a ideia de dois dos objetivos citados, dos quais destacam-se a procura de defeitos, mostra-se que, no teste feito em desenvolvimento (teste de componente, integração e de sistemas), o principal objetivo pode ser causar o maior número de falhas possíveis, de modo que os defeitos no software possam ser identificados e resolvidos.

A seguir serão apresentados os principais aspectos que impactam no desenvolvimento de um software e a relação desses pontos com o processo continuo de testes:

2. O teste de software estará diretamente ligado à qualidade final do produto.
3. A estruturação dos testes da maneira inadequada pode gerar um alongamento não planejado no prazo de desenvolvimento e consequentemente encarecer o custo do projeto com manutenções corretivas.
4. A equipe envolvida nos testes deve ser qualificada e ciente da sua importância no processo como um todo.
5. Seguindo esse raciocínio a implementação inadequada ou errada dos testes é um risco que sempre deve ser considerado. Ou seja, o escopo do projeto no todo tem como um de seus mais relevantes aspectos o teste bem planejado e executado.



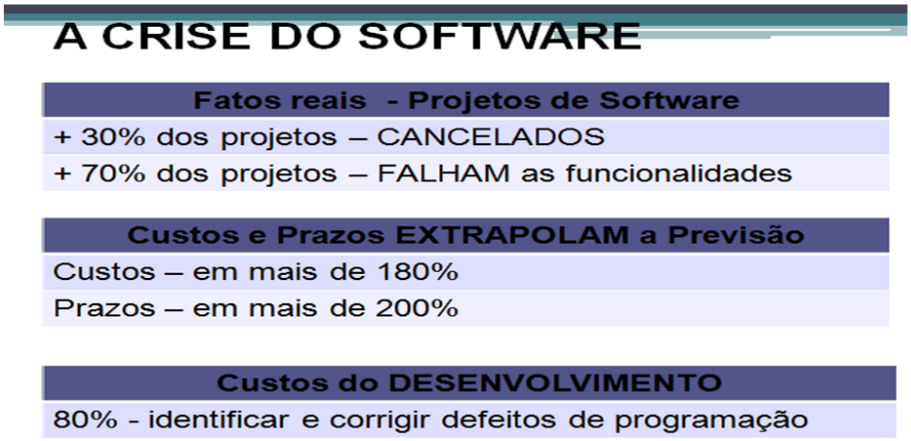
A CRISE DO SOFTWARE

A principio serão destacados os principais pontos de dificuldades relacionados ao desenvolvimento de softwares:

1. Projetos com orçamento bem acima do planejado
2. Projetos estourando o prazo
3. Software de baixa qualidade
4. Softwares não atendem os requisitos especificados
5. Código fonte complexo e difícil de ser manuseado
6. Projetos difíceis de serem gerenciados

O quadro a ser apresentado demonstra de maneira mais clara o impacto derivado de uma gestão inadequada no processo de planejamento, analise e desenvolvimento de um software.

Nesse contexto é vital a implementação de testes no software, de modo a garantir a confiabilidade e integridade do mesmo, trazendo o menor impacto possível no prazo e no custo de gerenciamento do projeto.



O QUE FOI A CRISE DO SOFTWARE?

A "crise do software" descreve um período com enorme dificuldades enfrentadas no desenvolvimento de software no fim da década de 60. A complexidade dos problemas, a ausência de técnicas bem estabelecidas e a crescente demanda por novas aplicações começavam a se tornar um problema sério. Foi nessa época, mais precisamente no ano de 1968, que ocorreu a Conferência da OTAN sobre Engenharia de Software (NATO Software Engineering Conference) em Garmisch, Alemanha. O principal objetivo dessa reunião foi estabelecer práticas mais maduras para o processo de desenvolvimento. Por essa razão, o encontro é considerado hoje como o nascimento da disciplina de Engenharia de Software.

Duas décadas depois, em 1986, Alfred Spector, presidente da Transarc Corporation, foi coautor de um artigo comparando a construção de pontes ao desenvolvimento de software. Sua premissa era de que as pontes normalmente eram construídas no tempo planejado, no orçamento, e nunca caíam. Na contramão, os softwares nunca ficavam prontos dentro do prazo e do orçamento, e, além disso, quase sempre apresentavam problemas.

Uma das consequências de tentarmos igualar metodologias de desenvolvimento de software a metodologias de engenharia tradicionais (como construção civil) é a visão de que o custo de modificar o programa aumenta exponencialmente ao longo do tempo.

A partir da metade dos anos 90, começaram a surgir discussões sobre se a forma faseada (cascata), era realmente o melhor modelo para o desenvolvimento de software. As metodologias ágeis de desenvolvimento surgiram exatamente para permitir maior flexibilidade no processo de desenvolvimento de sistemas e minimizar o custo de mudanças ao longo dos projetos.

CONSEQUÊNCIAS CRISE DE SOFTWARE

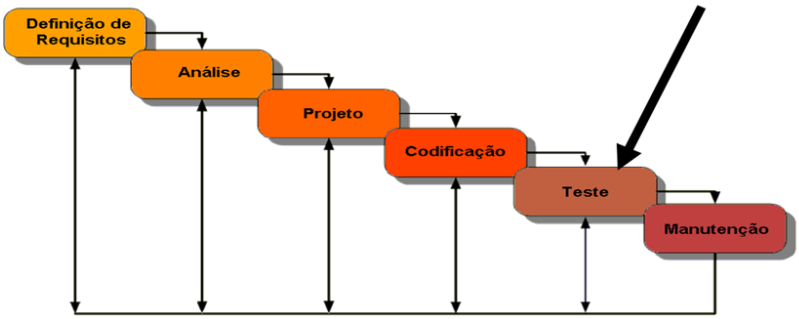
Uma das consequências da crise foi o surgimento de vários métodos e modelos de desenvolvimento, os quais devido a sua grande quantidade caracterizou-se por se tornar mais um problema frente a inexistência de um método genérico e confiável, outro aspecto esta também na aceitação de novas tendências, uma vez que devido a sua imaturidade no mercado eles dificilmente são incorporados devido a pouca confiabilidade que os desenvolvedores tem em relação aos mesmos.

Com tudo a maior consequência da crise foi o surgimento da engenharia de software o qual visa no desenvolvimento de tecnologias que facilitem e permitam a obtenção de softwares eficientes e baratos, sua importância o fez tornar-se um novo campo de pesquisa na computação.

SOLUÇÕES PARA A CRISE DE SOFTWARE

- O uso de melhores técnicas, métodos e ferramentas no processo de desenvolvimento
- Aplicação dos conceitos de Engenharia de software
- Mudança de paradigma sobre o que é desenvolver software e como deveria ser feito
- Melhor capacitação da equipe
- Priorizar os teste de software em relação as manutenções corretivas

CICLO DE VIDA DE UM SOFTWARE



Teste pode até ser a ultima, mas não a única atividade relacionada a qualidade de software.

ETAPAS DO CICLO DE DESENVOLVIMENTO DE SOFTWARE

1. **Definição dos requisitos**

Essa etapa consiste nas reuniões preliminares do projeto e levantamento das necessidades do cliente, além das tarefas de planejamento do projeto, tais como estimativas de gastos e despesas e elaboração de cronograma.

2. **Analise**

Esta etapa inclui a documentação e o estudo da viabilidade do projeto, pode ser vista como uma concepção de um produto de software e também como o início do seu ciclo de vida. Também é realizada a analise dos requisitos que foram especificados na fase anterior.

3. **Projeto**

O projeto do sistema é um processo de vários passos que se centraliza em quatro atributos diferentes do sistema: estrutura de dados, arquitetura do software, detalhes procedais e caracterização das interfaces. O processo de projeto representa os requisitos de uma forma que permita a codificação do produto (é uma prévia etapa de codificação).

4. **Codificação**

Esta é a etapa em que são criados os programas. É o momento onde os software são efetivamente desenvolvidos.

5. **Teste**

O processo de teste centraliza-se em dois pontos principais: as lógicas internas do software e as funcionalidades externas. Esta fase decide se foram solucionados erros de "comportamento" do software e assegura que as entradas definidas produzam resultados conforme os requisitos especificados.

6. **Manutenção**

Essa etapa consiste na correção de erros que não foram previamente detectados, em melhorias funcionais e de preferência e outros tipos de suporte. A etapa de manutenção à parte do ciclo de vida do produto de software e não pertence estritamente ao seu desenvolvimento.

Vantagens

- Permite controle departamental e gerencial.
- Oferece maior previsibilidade de prazos e custos: melhor planejamento e gerenciamento.
- Significativamente melhor que a abordagem casual no desenvolvimento do software.
- É adequado quando os requisitos são bem entendidos, como em aperfeiçoamento de um sistema existente.

Desvantagens

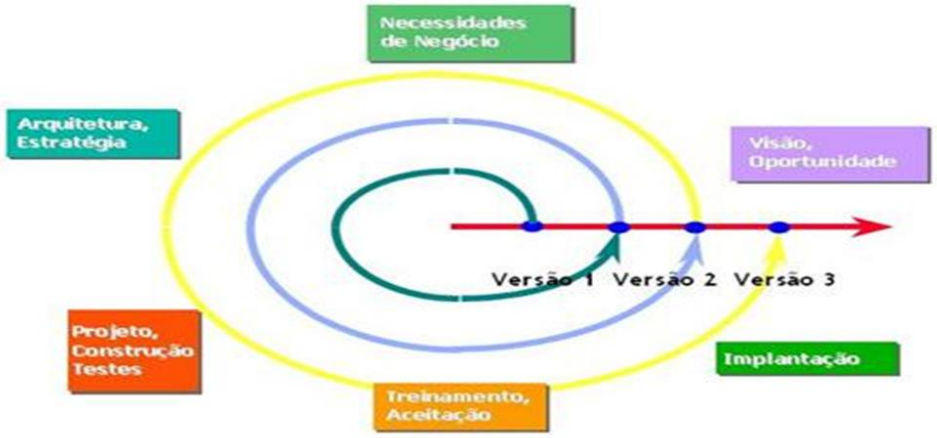
- Os projetos reais raramente seguem o fluxo sequencial que o modelo propõe
- Muitas vezes é difícil para o cliente declarar as exigências explicitamente.
- O ciclo de vida clássico exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de muitos projetos.
- Dificuldades em realizar mudanças com o processo em andamento – requisitos sempre mudam.
- O cliente deve ter paciência. Uma versão de trabalho dos programas não estará disponível até um ponto tardio do cronograma do projeto. Um erro crasso, se não for detectado até que o programa de trabalho seja revisto, pode ser desastroso.
- Estados de bloqueio: Membros da equipe ficam esperando outros membros terminarem sua parte.

Modelo Incremental

- Os requisitos iniciais do software são razoavelmente bem definidos, entretanto, devido ao escopo em geral do trabalho, o uso de um processo puramente linear não é utilizado.
- O modelo incremental combina elementos dos fluxos de processo lineares e paralelos.
- Este modelo aplica sequências lineares, de forma escalonada, à medida que o tempo vai avançando. Cada sequência linear gera incrementos (entregáveis/aprovados/liberados

- Vantagens
  - Entregas parciais facilitam a identificação e correção de erros entre os componentes do software.
  - Necessidades não especificadas nas fases iniciais podem ser desenvolvidas nos incrementos.
  - Cada iteração produz um conjunto de itens utilizáveis.
  - Os feedbacks de iterações anteriores podem ser usados nos próximos incrementos.
  - Os incrementos podem ser desenvolvidos por menos profissionais.
  - Entrega dos incrementos pemite o cumprimento do prazo especificado.
  - Facilita a manutenção dos "módulos".
- Desvantagens:
  - Número de iterações não pode ser definido no início do processo.
  - O fim do processo não pode ser previamente definido.
  - Gerenciamento e manutenção do sistema completo podem se tornar complexos.
  - Gerenciamento do custo é mais complexo devido ao número de iterações (verba pode acabar).

Modelo Espiral



O objetivo do modelo espiral é prover um metamodelo que pode acomodar diversos processos específicos. Isto significa que podemos encaixar nele as principais características de outras metodologias adaptando-os a necessidades específicas de desenvolvedores ou às particularidades do software a ser desenvolvido. Este modelo prevê prototipação, desenvolvimento evolutivo e cíclico, e as principais atividades do modelo cascata.

Sua principal inovação é guiar o processo de desenvolvimento gerado a partir deste metamodelo com base em análise de riscos e planejamento que é realizado durante toda a evolução do desenvolvimento. Riscos são circunstâncias adversas que podem surgir durante o desenvolvimento de software impedindo o processo ou diminuindo a qualidade do produto. São exemplos de riscos: pessoas que abandonam a equipe de desenvolvimento, ferramentas que não podem ser utilizadas, falha em equipamentos usados no desenvolvimento ou que serão utilizados no produto final, etc. A identificação e o gerenciamento de riscos é hoje uma atividade importantíssima no desenvolvimento de software devido à imaturidade da área e à falta de conhecimento, técnicas e ferramentas adequadas.

O modelo espiral descreve um fluxo de atividades cíclico e evolutivo constituído de quatro estágios:

- No estágio 1 devem ser determinados objetivos, soluções alternativas e restrições.
- No estágio 2, devem ser analisados os riscos das decisões do estágio anterior. Durante este estágio podem ser construídos protótipos ou realizar-se simulações do software.
- O estágio 3 consiste nas atividades da fase de desenvolvimento, incluindo design, especificação, codificação, testes e verificação. A principal característica é que a cada especificação que vai surgindo a cada ciclo deve ter sus verificação realizada apropriadamente.
- O estágio 4 compreende a revisão das etapas anteriores e o planejamento da próxima fase. Neste planejamento, dependendo dos resultados obtidos nos estágios anteriores - decisões, análise de riscos e verificação, pode-se optar por seguir o desenvolvimento num modelo Cascata (linear), Evolutivo ou Transformação.

1. Ambiente e Configuração dos Testes

Nesta sessão são descritos o ambiente de testes e as configurações necessárias para a execução deles.

1.1 Plataformas

Os testes devem ser executados em todas as plataformas suportadas pelo software, pois é necessária uma documentação que sustente essa garantia de suporte dada pelo fabricante do software. Cada uma das plataformas a serem usadas nos testes deve ser descrita com detalhe, com foco nas características mais relevantes ao software a ser testado: tipo de máquina, capacidade de processamento, sistema operacional, quantidade de memória disponível, além de outras informações relevantes ao equipamento que trabalhará com seu software/aplicativo. Os usuários do seu aplicativo provavelmente irão instalá-lo ou executá-lo em computadores com amplas diferenças, como versões de sistemas operacionais e navegadores da Web diferentes, por isso certifique-se de submeter o software em diferentes plataformas.

1.2 Configurações

Assim como as plataformas, as configurações usadas em cada ambiente de testes devem ser descritas detalhadamente: softwares relacionados àquele a ser testado, drivers necessários ao seu uso, entre outras. Uma boa opção é usar parâmetros quando se deseja que um teste seja executado com dados de teste diferentes, pois é fácil definir parâmetros diferentes para casos de teste diferentes. As configurações de teste são melhores para variações na plataforma de hardware ou de software em que o aplicativo no teste é instalado.

2. Analise de teste

A análise dos testes são descritas minuciosamente nesta sessão, que abrange os objetivos, as dependências, a preparação e finalmente os casos de teste.

2.1 Objetivos dos testes

Os objetivos dos testes são pautados pelas características finais desejadas do software: segurança, integração com outros processos ou softwares, saídas conforme requisitos especificados e geração de determinados dados são alguns exemplos.

Cada caso de teste possui suas características especificas, suas aplicações e sua importância para a integridade, usabilidade e eficiência do produto final.

Dentre os mais variados objetivos vale apena destacar os seguintes:

- Exercitar as estruturas de dados internas para garantir a sua validade.
- Exercitar todas as decisões lógicas em seus lados verdadeiro e falso;
- Validar todos os ciclos nos seus limites e dentro dos intervalos operacionais,
- Buscar erros de inicialização e término.
- Buscar erros de comportamento
- Analisar o comportamento do software em situações excepcionais
- Validar o tempo de resposta do sistema

Exemplo de Objetivos incorporados ao Teste de software

<u>Estruturação padrão, baseada nas condições, abordagem e critérios de aceitação:</u>

Condição do teste: Condição que deve ser atendida para que o objetivo seja satisfeito

Abordagem: De que forma esse objetivo pode ser avaliado?

Critério de aceitação: Critério que define se o objetivo foi atendido.

2.1.1 Objetivo 1—Segurança

Condição do teste: Somente usuários autorizados devem ter acesso às funcionalidades do software.

Abordagem: Devem ser usadas combinações corretas e incorretas/inválidas de usuário e senha na tela de autenticação do usuário.

Critério de aceitação: O processo de autenticação para uso do software deve negar acesso em caso de combinações incorretas de usuário e senha e permitir o acesso quando a combinação fornecida for correta

2.1.2 Objetivo 2 —Desempenho

Condição do teste: Processo de autenticação deve ser realizado com plausível.

Abordagem: Devem ser usadas combinações corretas e incorretas/inválidas de usuário e senha na tela de autenticação do usuário de modo a medir e avaliar o tempo de resposta do sistema nesses casos

Critério de aceitação: O processo de autenticação para uso do software deve negar acesso ou permitir o acesso dentro de um limite de tempo estipulado.

2.1.2 Objetivo 3 —Usabilidade

Condição do teste: Verificar se as funções da aplicação são claras e facilmente entendidas e operadas pelos usuários.

Abordagem: Deve se verificar se a navegação através das telas reflete as regras de negócios e requisitos, incluindo campos, janelas, métodos de acesso bem como o entendimento do usuário das telas e respostas do software.

Critério de aceitação: A interface deve fornecer ao usuário a navegação apropriada e com a maios simplicidade possível.

2.2 Dependência dos casos de testes

Nessa sessão são descritos os requisitos que devem ser atendidos para a realização dos testes. Pode ser necessária a preparação de dados de entrada para que o software os reconheça, por exemplo.

Frequentemente temos que testar a interação entre parâmetros dependentes de um sistema. Se montarmos um conjunto contendo todas as possíveis combinações de parâmetros e criarmos um caso de teste para cada elemento desse conjunto, provavelmente serão encontrados todos os bugs, falhas e erros, entretanto o esforço, o tempo gasto, e os custos seriam enormes.

Para resolver este problema temos que criar casos de teste não usando o conjunto original, mas sim um subconjunto deste.

Para isso é interessante utilizar técnicas de testes especificas que busquem validar os parâmetros e ao mesmo tempo minimizem os esforços alocados.

2.3 Preparação para os casos de testes

Uma parte muito importante que deve ser feita antes de efetuar os testes de softwares é a preparação de todo o ambiente e cenário em que serão executados os testes. Cada alteração efetuada no ambiente, cada atualização ou downgrade de quaisquer tipos de ferramentas ou softwares que estão relacionados direta ou indiretamente ao funcionamento da aplicação devem ser previamentes configuradas corretamente antes da execução dos testes. Essas informações devem estar documentadas e descritas em detalhes nos casos de testes, focando nas características mais relevantes e que possam causar impactos durantes a realização do caso de teste. Por exemplo: Sistema operacional, modelo da maquina utilizada para teste, modelo do processador, além de toda a configuração. Lembrando que para cada tipo de teste, as necessidades de informações podem variar.

Exemplo:

Sistema Operacional: Windows 10 build 10576

Sistema instalado versão 6.20

Browser: Mozila Firefox versão 10.4.56.0

2.4. Casos de teste

Com o documento de especificação do design de teste pronto, parte-se para a criação dos casos de teste e dos procedimentos de teste. O Modelo de Especificação de Casos de Teste do IEEE 829 inclui a descrição do ambiente de testes, as pré e pós condições do teste, a descrição do teste e seus objetivos, os inputs e outputs esperados e as dependências entre os testes. Ele é um documento bem detalhado, que poderá ser usado como referência em todas as etapas do teste e pode sofrer atualizações para melhor se adequar ao design de testes e ao comportamento que o software apresentar durante os testes. Por fim, a Especificação dos Procedimentos de Teste é definida pelo Padrão IEEE 829 como o documento que reúne o propósito dos testes, seus requerimentos específicos e finalmente o passo a passo detalhado de realização dos testes. É um documento ainda mais técnico que os demais, que descreve com detalhes e em sequência os passos a serem seguidos em cada caso de teste, seja ele manual ou automatizado. Ele também é chamado de script de testes e, juntamente com os demais documentos descritos nesse capítulo, faz parte do modelo de Plano de Teste explorado no capítulo seguinte.

Passo a passo do design do Plano de Teste de software

Informações sobre o documento

Versão	Data	Descrição	Responsável
0.1	*DD/MM/AAAA*	*<Primeiro esboço do documento criado>*	*Nome*
0.2	*DD/MM/AAAA*	*<Documento revisado e editado>*	*Nome*
1.0	*DD/MM/AAAA*	*<Aprovado>*	*Nome*

Sumário de alterações do Documento

Versão	Data	Descrição	Responsável
0.1	*DD/MM/AAAA*	*<Primeiro esboço do documento criado>*	*Nome*
0.2	*DD/MM/AAAA*	*<Documento revisado e editado>*	*Nome*
1.0	*DD/MM/AAAA*	*<Aprovado>*	*Nome*

Sumário de alterações do Documento

Versão	Data	Descrição	Responsável
0.1	*DD/MM/AAAA*	*<Primeiro esboço do documento criado>*	*Nome*
0.2	*DD/MM/AAAA*	*<Documento revisado e editado>*	*Nome*
1.0	*DD/MM/AAAA*	*<Aprovado>*	*Nome*

Aprovadores

Nome	Cargo
Nome	*<Gerente de Qualidade>*
Nome	*<Arquiteto de Software>*
Nome	*<Gerente de Projetos>*

Documentos de Referência

Documento	Referência
<Plano do projeto>	*Fonte*
<Especificação dos requerimentos>	*Fonte*
<Design de alto nível>	*Fonte*

2.4.1 Casos de teste 1 - Acesso ao sistema

Caso de Teste 1: Acesso ao sistema
Pré-Condições/Condição do Teste: 1.Sistema devidamente instalado no computador. 2.Usuário cadastrado no sistema.
Abordagem/Passos: 1.Executar a aplicação 2.Preencher o campo usuário com um usuário cadastrado no sistema 3.Digitar a senha do usuário 4.Clicar em OK
Critérios de Aceitação/ Resultados Esperados: 1.Acesso ao sistema realizado com sucesso. 2.O sistema deverá exibir a tela principal de operação.

- Condição/Descrição: O caso de teste abaixo tem como objetivo verificar se um o acesso ao sistema ocorre conforme previsto.

- Tempo estimado: Para a execução desse teste, estima-se um tempo médio de três minutos para execução. Sendo que a autenticação e acesso a sistema deve levar até 5 segundos.

- Critério de saída/Resultado esperado: O acesso ao sistema deve ser concluído como previsto, seguindo as especificações de tempo, segurança e desempenho.

- Requisitos do ambiente: O software a ser analisado deve estar instalado no ambiente de testes a ser utilizado.

- Objetivos mapeados: Esse teste tem como objetivo verificar a Segurança do sistema, validando a autenticação do usuário bem como garantir a usabilidade da tela de acesso ao sistema.

2.4.2 Casos de teste 2 - <Senha Inválida>

Caso de Teste 2: Senha inválida
Pré-Condição: 1.Sistema devidamente instalado no computador. 2.Usuário cadastrado no sistema.
Passos: 1.Executar a aplicação 2.Preencher o campo usuário com um usuário cadastrado no sistema 3.Digitar uma senha inválida 4.Clicar em OK
Resultados Esperados: 1.O sistema deverá informar que o usuário ou a senha estão incorretos. 2.O sistema não deverá permitir o acesso.

- Condição/Descrição: O caso de teste apresentado, consiste em verificar se um usuário consegue acessar ao sistema mesmo com um senha inválida.

- Tempo estimado: Para a execução desse teste, estima-se um tempo médio de cinco minutos para execução. Sendo que a autenticação e acesso a sistema deve levar até 3 segundos.

- Critério de saída/Resultado esperado: O acesso deve ser negado e uma mensagem exibida na tela do sistema informando que o usuário informou uma senha inválida.

- Requisitos do ambiente: O software a ser analisado deve estar instalado no ambiente de testes a ser utilizado.

- Objetivos mapeados: Esse teste tem como objetivo verificar a Segurança do sistema, validando a autenticação do usuário com a senha digitada

2.4.3 Casos de teste 3 - <Usuário Inválido>

Caso de Teste 3: Usuário inválido
Pré-Condição: 1. Sistema devidamente instalado no computador. 2. Usuário cadastrado no sistema.
Passos: 1. Executar a aplicação 2. Preencher o campo usuário com um usuário inválido 3. Digitar uma senha 4. Clicar em OK
Resultados Esperados: 1. O sistema deverá informar que o usuário ou a senha estão incorretos. 2. O sistema não deverá permitir o acesso.

- Condição/Descrição: O caso de teste abaixo, consiste em verificar se um usuário não autorizado consegue acessar as funcionalidades do software.
- Tempo estimado: Para a execução desse teste, estima-se um tempo médio de cinco minutos para execução. Sendo que a verificação de autenticação deve ser efetuada pelo sistema em até 3 segundos.
- Critério de saída/Resultado esperado: O acesso deve ser negado e uma mensagem do sistema informando que o usuário não possui autorização para aquele acesso, deve ser exibida.
- Requisitos do ambiente: O software a ser testado deve estar instalado no ambiente de testes. Possuir um usuário cadastrado no sistema.
- Objetivos mapeados: Esse teste tem como objetivo verificar a Segurança do sistema, validando a autenticação de usuário e senha.

Cuidados ao escrever casos de teste

Os casos de testes são uma excelente forma de executar os testes de forma segura e tendo uma visão de um resultado esperado. Entretanto, alguns cuidados devem ser tomados com o nível de detalhamento e com a complexidade dos casos de testes.

- Não tente verificar todos os comportamentos de uma da função em um único caso de teste
- Tentar atacar por aspectos, cenários, contextos, etc
- Não confunda a configuração de uma prioridade de um software com o teste daquela prioridade.
- O cérebro humano é limitado. Crie sempre um oráculo para ajudar.

3.Execução dos testes

Essa sessão lista as funcionalidades do teste de software, bem como sua importância e suas funcionalidades, além de apresentar as principais definições de teste, erro, falha e defeito. Outros aspectos a serem apresentados são as Abordagens, níveis, categorias e estratégias ligadas ao teste de software.

3.1 Teste de Software

O QUE É TESTE ?

De uma maneira geral, pode-se definir os testes como: O processo de demonstrar que os defeitos não estão presentes. E ainda: Demonstrar que os sistemas e programas funcionam corretamente e fazem o que deveriam fazer.

Em um aspectos mais amplo as definições apresentadas estão incompletas, pois enxergam os testes somente com uma forma de comprovar que tudo funciona conforme o que foi estabelecido. Todas as definições dão uma dimensão positiva, querem provar que algo funciona.

POR QUÊ?

Porque quando queremos provar que algo funciona idealizamos um conjunto de cenários favoráveis a utilização do software. Por outro lado, quando queremos provar que algo não funciona, incluímos não só os cenários positivos, mas também os cenários negativos.

Os benefícios esperados com a implantação de um processo de teste são inúmeros destacando dentre eles a obtenção de maior qualidade dos softwares produzidos pelas empresas além de redução do tempo e capital empregado com manutenções corretivas. Informações como a quantidade de defeitos por versão de produto, quanto do software foi testado, quantidade de ciclos entre o desenvolvimento e o teste auxilia não só no melhor entendimento dos processos internos como também aumentam a confiança da equipe no produto. Além dessas contribuições, o panorama que o teste expõe traz a luz questões outrora não conhecidas e que são pertinentes para a tomada de decisões .

O QUE NÃO É TESTE DE SOFTWARE

Dentre inúmeras definições do que não é considerado teste de software, destacam-se:

1. Teste de software não deve ser considerado um processo burocrático.
2. Uma atividade limitada a seguir um roteiro
3. Uma atividade que possa ser desempenhada por qualquer pessoa

3.2 Definições de teste, erro, falha e defeito

CONCEITOS INICIAIS

- Erro: é a diferença entre o valor esperado e o valor obtido.
- Falha: é a produção de resultado, algo incorreto em relação a especificação.
- Erro humano ou engano: é a atividade humana executada de forma equivocada, resultando em falhas de software.
- Defeito: é a representação no software , do erro humano obtido.

Fonte: IEEE Std 729, *Standard Glossary of Software Engineering Terminology*

Objetivos do Teste

- Encontrar erros ou expor falhas.
- Verificar que um software atende a um determinado requisito.
- Encontrar divergências entre o que o software faz e o que foi especificado.
- Ganhar confiança sobre o nível de qualidade e prover informações.

3.3 Abordagens, níveis, estratégias e categorias

Abordagens

- Teste Funcional (caixa preta): é um método em que o testador deriva casos de teste baseados nos elementos e requisito de teste, que por sua vez foram obtidos das Especificações de Requisitos. Esse tipo de teste é usado quando não tem acesso ao código conte do programa, nesse caso o foco do teste é na validação dos requisitos funcionais.
- Teste estrutural (caixa branca): é um método em que o testador deriva os casos de teste a partir da estrutura interna do software. Esse método se concentra nas estrutura interna do software, avaliando suas decisões lógicas, seus loops, a estrutura interna de dados e o caminho dentro dos módulos.

Níveis de teste

- Teste de unidade: verifica se cada unidade funciona de acordo com o que foi especificado
- Teste de integração: montagem (projeto/arquitetura) do software e verificação da interface das unidades.
- Teste de aceitação: verifica se o software satisfaz os requisitos do cliente
- Teste de sistema: Verifica todos os demais requisitos da aplicação

Estratégias

- Teste de progressão:

Novos casos de testes criados à medida que o software ganha novas funcionalidades.

Seu objetivo é verificar o funcionamento das inovações realizadas

- Teste de regressão:

Re-execução de um sub-conjunto de testes (parcial ou total) previamente executados.

Seu objetivo é assegurar que as novas alterações não afetaram outras partes do produto que já funcio-

Uma estratégia de testes de software descreve a abordagem geral e os objetivos das atividades de teste. Ela deve contemplar os níveis ou fases de teste, os tipos de testes a serem realizados e as técnicas para sua execução. A estratégia de testes de softwares também deve descrever com clareza os critérios para a conclusão dos testes e os critérios de sucesso a serem usados.

Responsáveis: Gerente de Projeto, Engenheiros de Software e Especialistas em Testes.

Importância: A importância das estratégias é evitar tempo desperdiçado, esforço desnecessário, infiltração de erros sem serem descobertos.

Passos: Primeiramente os testes focalizam um único componente ou um pequeno número de componentes relacionados, aplicados para descobrir erros nos dados e na lógica de processamento. Em seguida testes de alto nível são executados para descobrir erros na satisfação dos requisitos dos clientes. E por fim os erros encontrados devem ser diagnosticados e corrigidos usando o processo de depuração.

Produtos: Especificação de Teste – documenta a abordagem da equipe de software para o teste, definindo o Plano de Testes – descreve uma estratégia global e um procedimento que define passos específicos de teste e os testes que serão conduzidos.

Estratégias de Teste de Software para arquitetura de Software Convencionais

O quê	Quem	Como
Teste de Unidade – Código	Desenvolvedores	Focaliza cada componente individualmente, garantindo que ele funcione adequadamente como uma unidade. Usa técnicas de testes que exercitam caminhos específicos na estrutura de controle de um componente, para garantir completa cobertura e máxima detecção de erros. Teste Caixa-Branca.
Teste de Integração – Projeto e Arquitetura de Software	Desenvolvedores	Os componentes devem ser montados ou integrados para formar o pacote de software completo. Cuida dos tópicos associados com os problemas duais de verificação e construção de programas. As técnicas de projeto de casos de teste que enfocam as entradas e saídas são mais prevalentes durante a integração.
Validação – Sw construído x Análise de Requisitos (alto nível)	Analista de Teste	Fornecem garantia final de que o software satisfaz a todos os requisitos funcionais, comportamentais e de desempenho.
Sistema	Analista de Teste	Verifica se todos os elementos (hardware, pessoal, banco de dados) combinam adequadamente e se a função/desempenho global do sistema é alcançada.

Categorias de teste

TESTE DE USABILIDADE

Este teste tem por objetivo simular a utilização do software na perspectiva o usuário final.

Verifica-se a facilidade de navegação, clareza dos textos e mensagens, facilidade de aprendizado , padronização visual entre outros aspectos.

TESTE DE INTEGRAÇÃO

Esse Estágio garante que os componentes funcionam corretamente ao serem combinados, verifica as dependências entre os módulos. Permite a condução de testes para detectar erros relacionados a interface paralelamente a construção da aplicação. O objetivo é encontrar falhas provenientes da integração interna dos componentes de um sistema, geralmente encontradas em caso de envio e recebimento de dados. É possível que dados sejam perdidos através de uma interface; um módulo pode ter um efeito imprevisto ou adverso sobre outro; subfunções quando combinadas podem não produzir o resultado esperado; uma imprecisão aceitável individualmente pode em conjunto tornar-se inaceitável.

TESTE DE REGRESSÃO (ESTRATÉGIA)

Essa é uma técnica de teste aplicável a uma nova versão de software ou à necessidade de se executar um novo ciclo de teste durante o processo de desenvolvimento. Consiste em se aplicar, a cada nova versão do software ou a cada ciclo, todos os testes que já foram aplicados nas versões ou ciclos de teste anteriores do sistema. Inclui-se nesse contexto a observação de fases e técnicas de teste de acordo com o impacto de alterações provocado pela nova versão ou ciclo de teste. Para aumento de produtividade e de viabilidade dos testes, é recomendada a automatização do processo de teste.

TESTE DE RECUPERAÇÃO

Essa categoria de teste atua forçando o sistema a falhar, em relação à diversos aspectos do sistema computacional e verificar a resposta (em termos de recuperação do sistema)

TESTE DE INSTALAÇÃO

Esse tipo de teste é utilizado para validar os procedimentos de instalação de uma aplicação, tais como tempo decorrido e uso de memória, esse teste tem como função avaliar os procedimentos de instalação previstos possibilitam as várias alternativas especificadas nos requisitos identificados

TESTE FUNCIONAL

Esse teste atua verificando se os requerimentos, regras de negócio e funcionalidades do sistema presentes na documentação foram implementadas com sucesso. Valida as funcionalidades descritas na documentação, é um teste muito importante uma vez que irá avaliar se o software entrega o que foi prometido.

TESTE DE INTERFACE

É um modelo de teste utilizada para verificar se a navegabilidade e os objetivos da tela funcionam como especificados e se atendem da melhor forma ao usuário. Atua visando simplificar o entendimento do usuário da interface (telas e opções) do sistema.

TESTE DE PERFORMANCE

É um modelo de teste projetado para testar o desempenho do software durante a sua execução em um contexto de sistema integrado. Busca descobrir situações que levam à degradação e possível falha do sistema. Ocorre implicitamente ao longo de todo o processo de teste, mas o verdadeira avaliação da performance de uma aplicação somente pode ser avaliado após a plena integração de todos os seus componentes. Frequentemente são integrados aos testes de estresse e dependem da utilização de ferramentas específicas para sua execução.

TESTE DE CARGA

Nessa situação o banco de dados é carregado com informações ou então é maximizado o número de transações concorrentes para então verificar o comportamento da aplicação. Observa-se se o software continua funcionado corretamente com cargas diferentes de trabalho. Esse teste é utilizado para verificar o comportamento da aplicação quando submetida a variações de carga de trabalho, medindo e avaliando seu desempenho e os impactos dessa alta exigência do sistema.

TESTE DE ESTRESSE

O teste de estresse submete o sistema a situações anormais. O programa deve ser executado de forma que demande recursos em quantidade, frequência ou volume anormais. Baseia se em testar os limites do software e avaliar seu comportamento, verificar até quando o software pode ser exigido e quais são as falhas decorrentes do teste.

TESTE DE CONFIGURAÇÃO

Esse modelo de teste atua verificando se os requisitos que estão sendo testados funcionam em diferentes configurações de software e hardware.

TESTE DE SEGURANÇA

Os testes de segurança permitem avaliar as vulnerabilidades em aplicações e serviços frente a diferentes tipos de ataques de segurança, como por exemplo: ataques de negação de serviço e descobrir novas vulnerabilidades antes que sejam exploradas por atacantes. Atua também verificando se o software se comporta adequadamente mediante a tentativas ilegais de acesso, Para isso, testa se todos os mecanismos de proteção embutidos na aplicação de fato a protegerão de acessos indevidos.

TESTE DE CONFIABILIDADE E DISPONIBILIDADE

É uma modalidade de teste que consiste no monitoramento de um software por um determinado período do tempo, verificando se mantem as informações acessíveis e avaliando o nível de confiabilidade da arquitetura do solução

3.4 Casos de teste e massa de dados

Casos de teste

Um Caso de teste é um conjunto de entradas e saídas esperadas. Desenvolvido para verificar ou validar um objetivo particular de um aplicação.

Uma entidade identificável e associada a "algum comportamento" esperado do software. Possui um conjunto e entradas e uma lista de saídas esperadas.

Massa de Dados

Dados de Entrada levam ás Massas de Testes

A massa de dados pode ser genérica, mas deve ser sensível ao contexto do caso de teste.

Pontos importantes em relação a massa de dados:

- Meta-dados (dados sobre os dados)

Dicionários de Dados

Casos de teste e objetos de teste envolvidos

- Podem ser fictícios ou reais. Exemplo:

1. Arquivos texto para importação gerados automaticamente (fictício)
2. Base de dados do cliente (dados reais)

Oráculo

O oráculo é utilizado para apontar adiantadamente, a saída esperada em um caso de teste. Por exemplo, em um caso de testes que envolva cálculos, utilizar uma planilha de Excel com as fórmulas realizando um determinado cálculo e exibindo um resultado esperado e comparando-o com resultado do sistema.

Por que fazer casos de teste?

Os casos de testes são uma forma metódica de executar os testes, com o intuito de executar um teste com alta repetitividade, utilizando-se de uma documentação consistente, afim de se obter uma melhor transferência de conhecimento e elevar o nível de trabalho realizado..

3.5 Técnicas de Design do teste

O design do teste descreve as condições que os testes devem atender e o comportamento que se deve esperar do software a ser testado. Ele pode ser baseado nas requisições do sistema, nas necessidades de negócio ou no fluxo de processos para os quais o software foi projetado. Nesse documento são identificados os atributos a serem testados, as abordagens que serão utilizadas, os testes em si e os critérios de aprovação e reprovação de cada um deles.
O design de testes, portanto, é a base para a criação dos demais documentos, pois define o escopo do plano de testes, o que será abrangido e o quais os resultados esperados. Ele é definido pelo padrão IEEE 829 no Modelo de Especificação de Design de Teste.
Há três técnicas para design de testes:

1. Caixa-preta (ou baseado em especificações)
2. Caixa-branca (ou baseado na estrutura do software)
3. Baseado na experiência de uso de softwares semelhantes

3.5.1 Caixa-preta

O design de testes que se baseia nas especificações do software (caixa-preta) foca em sua funcionalidade, verifica se ele cumpre o que foi proposto. Geralmente são definidos dados de entrada e os esperados dados de saída. Ao final de cada ciclo de testes, os dados de saída são comparados com aqueles definidos no Plano de Testes e a aprovação depende da paridade entre eles. A técnica da caixa-preta é útil para todos os níveis de teste, desde os de componentes até os de sistema e aceitação, pois é fundamental que o software atenda suas especificações. Um exemplo de ferramenta de suporte ao teste de caixa-preta é o Rational Functional Tester (RFT).

3.5.2 Caixa-branca

Já o design baseado na estrutura (caixa-branca) investiga o funcionamento do software e analisa sua estrutura interna, por isso também é chamado de caixa-branca ou de vidro. Os testes podem se restringir a componentes específicos ou abranger o software como um todo, mas a abordagem depende dos processos executados pelo software. Uma prática comum é escolher processos críticos do software e executá-los de várias maneiras diferentes, com o objetivo de detectar falhas e medir sua performance em cenários distintos. Esse tipo de técnica também é útil em todos os níveis de teste, mas com uma abordagem diferente dependendo do nível. Para testes de componentes e integração, a estrutura do software é o foco; em testes de sistema e aceitação, a estrutura relativa ao uso do software (como menus e componentes principais) se torna o alvo dos testes. A IBM possui, dentro da suite Rational, a ferramenta Rational Purify que é um depurador em tempo de execução, baseada na técnica de análise estática de código-fonte. Além disso, algumas funcionalidades que viabilizam testes do tipo caixa-branca podem ser encontradas embutidas no Rational Software Architect e no Rational Application Developer.

3.5.3 Baseado na experiência e conhecimento

Por fim, o design de testes baseado na experiência usa o conhecimento e a intuição do engenheiro de testes e usuários experientes para a determinação das condições de teste e a criação de casos de testes. Engenheiros de teste experientes na área do software a ser desenvolvido têm mais facilidade em imaginar cenários em que o produto possa apresentar vulnerabilidade. E o envolvimento de usuários enriquece o Plano de Testes, pois eles têm outra perspectiva do software e das necessidades que ele deve atender. Para testes de sistemas de baixo risco essa técnica costuma ser a única usada. Nos demais casos, ela é complementar às demais, e muito útil quando não há uma especificação de software bem definida. Neste caso é possível valer-se de ferramentas de gerenciamento de ciclo de vida e defeitos, como o Rational Team Concert e o Rational Quality Manager, que fornecem informações valiosas sobre o histórico do desenvolvimento do software. Essas informações podem ser utilizadas para a criação de casos de teste focados nas áreas críticas onde houve maior volume de defeitos, por exemplo. É possível também fazer comparações entre versões diferentes de um mesmo software sob a perspectiva do desempenho, utilizando o Rational Performance Tester.

3.6 Cobertura

Essa sessão lista as funcionalidades do software que são cobertas pelos testes do Plano, além de outros aspectos, como usabilidade, performance e segurança. Ela não deve ter um caráter técnico, e sim deve ser feita sob o ponto de vista do usuário, abordando o que interessará a ele e refletindo o que será testado através dos casos de teste.

Noções Básicas

Critérios para teste se apoiam em alguns conceitos básicos:

- **Cobertura de comandos:** um parâmetro de quantas linhas do programa são verificadas em um teste.
- **Cobertura de decisão:** é uma noção de quantos desvios são verificados por um teste.
- **Cobertura de caminho:** especifica, dos caminhos possíveis de um software, quantos são cobertos na execução de um teste.

Adequação à mutação: analisa quanto o teste é sensível à inserção de falhas artificiais no software. Um critério de teste é o que define quais propriedades precisam ser testadas para garantir inexistência de erros. Como é impossível garantir inexistência, o conceito é utilizado, na prática, para definir uma qualidade mínima que será validada pelo teste.

Categorias de Critérios

Podemos dividir os critérios de diversas formas: pela fonte da informação usada para especificar o teste, pelo uso exclusivo da interface para gerar os testes, e pela forma com que o teste é realmente conduzido. Utilizando esta última forma para classificar os critérios, o artigo faz uma análise de cada uma.

Critérios Baseados em Estrutura

Critérios baseados na estrutura do software especificam o teste em termos da cobertura de um conjunto de elementos do software ou especificação. Estes critérios podem ser, por sua vez, divididos em dois grupos:

Baseados no Programa: estes incluem critérios de adequação de controle de fluxo, que analisam o grafo de execução de um programa; critérios de adequação baseados no fluxo de dados, que analisam os caminhos que o código percorre entre a inicialização dos dados e os pontos onde são alterados e lidos; e critérios de adequação baseados em uma combinação dos dois tipos, que usam relações de dependência entre os comandos e os seus dados operandos.

Baseados na Especificação: além de permitir a verificação da corretidão das saídas de um programa, a especificação pode ajudar a escolher casos de teste e medir a adequação dos critérios de teste. Estes critérios focam na especificação, e ignoram o programa em si. Há duas formas básicas se especificar formalmente um software: uma, baseada em modelo, como especificações escritas em Z ou VDM, e outra baseada em propriedades algebraicas, que especificam quais equações as operações do programa devem obedecer.

Critérios Baseados em Falhas

Testes baseados em falhas focam em detectar falhas no software: critérios de teste baseados em falhas utilizam uma medida da capacidade destes testes efetivamente encontrarem uma falha. Há uma série de formas diferentes de se aplicar estes testes e estabelecer critérios:

- **Introdução de Erros:** esta técnica envolve introduzir erros no software e executar o teste, amostrando dos erros resultantes quantos provieram das falhas introduzidas. As falhas em geral são inseridos manualmente.

- **Teste de Mutação:** envolve criar uma coleção de versões do software que diferem entre si e aplicar a bateria de testes. A amostragem da saída dará uma proporção de falhas em relação ao original.

- **Teste de Perturbação:** este método foca em considerar falhas que ocorrem em um um espaço de erros, e gerar funções de perturbação para este programa; são posteriormente executados os variantes perturbados e analisados os erros encontrados.

- **O modelo RELAY:** faz comparação entre o software com falhas e uma variante hipoteticamente perfeita, e localiza as menores unidades possíveis do programa que possuem uma falha; é feito, então, uma análise iterativa destas regiões sensíveis.

- **Teste de Mutação de Especificação:** são implantados falhas na especificação, e após execução de testes no software, são verificados quais erros provieram das alterações introduzidas.

Critérios Baseados em Erros

Estes critérios se focam em analisar o teste do programa em certos pontos propensos a erros de acordo com nossa experiência. Se apóia na idéia de particionar o domínio de entrada e saída e obter regiões equivalentes para o teste.

- **Partição de Entrada baseado na Especificação:** a partir da especificação, são determinadas regiões de equivalência; estas são em geral separadas manualmente, embora tentativas existam de automatizar o processo.

- **Partição de Entrada baseado no Programa:** analisando o resultado do processamento de diferentes partes do programa, podem ser determinados subdomínios equivalentes; em geral, estes equivalem aos caminhos do programa.

- **Análise de Barreira:** se concentra em selecionar N casos de teste nas fronteiras de um espaço de entrada, e um caso de teste externo a esta fronteira. O objetivo é determinar se as barreiras são suficientes. Análise Funcional: enfatiza a integridade *dentro* de cada subdomínio; deve ser usada para complementar a Análise de Barreira e não isoladamente.

Uma Comparação entre Critérios de Adequação

- **Habilidade de detecção de Falhas** É uma medida bastante direta da efetividade de um critério de teste. É feita de algumas formas diferentes: fazendo experimentos estatísticos em cima dos dados dos testes, executando simulações dos testes em si com entradas controladas, e fazendo análise formal dos relacionamentos entre critérios de adequação, provando que a relação vale ou não para todos os critérios usados.

- **Confiabilidade do Software** É difícil fazer uma análise objetiva da confiabilidade do software diretamente. Existem pesquisas na área que concluem que é mais difícil de se obter certeza de confiabilidade usando teste randômico do que teste de partição.

- **Custo do Teste** Existem experimentos realizados analisando o custo do testo com caminhos não-executáveis, e há evidencias de que verificar o custo de testar através do número de casos de teste pode ainda vir a fornecer uma imagem do custo total.

Verificação Axiomática de Critérios

Além das formas apresentadas para avaliar o uso dos critérios descritos, existe uma terceira forma de estudar as propriedades do critério. Esta forma se concentra em estabelecer as propriedades fundamentais da adequação de teste, e verificar se estas propriedades são verificáveis para cada critério.

4. TestLink

O **TestLink** é um software web desenvolvido para teste de software que visa facilitar testes e assegurar a qualidade de software. Foi desenvolvido e mantido por varias equipes ao longo de sua existências. A plataforma oferece suporte para caso de teste, plano de teste, teste de unidade entre outros, também conta com suporte para relatórios e estatísticas. Possui como grande atrativo, ser acessado pela web, o que facilita a execução de testes por diversas equipes que estão separadas fisicamente.

Objetivos do TestLink

- ◊ Editar e organizar os casos de teste
- ◊ Facilitar o planejamento dos testes funcionais
- ◊ Facilitar o registro da execução dos testes
- ◊ Facilitar a execução dos testes de software em times

Tipos de Conta

O Testlink trabalha com permissões associadas a tipos de conta que os usuários podem utilizar, portanto após o primeiro login é recomendável que criemos as contas necessárias para o projeto e de-sabilitemos a conta padrão. Os seguintes tipos de conta são possíveis:

Guest: só tem permissão para visualizar relatórios;

Tester: só pode documentar a execução dos Casos de Testes;

Test Senior: pode documentar a execução de Casos de Testes, manipularem a área de Especificação de Testes criando Suíte de Testes ou Casos de Testes.

Leader: possui permissão para gerenciar Planos de Testes, criarem Builds, definir prioridades e todas as permissões dos usuários guest, tester e test senior.

Administrator: possui todas as permissões dentro do Testlink;

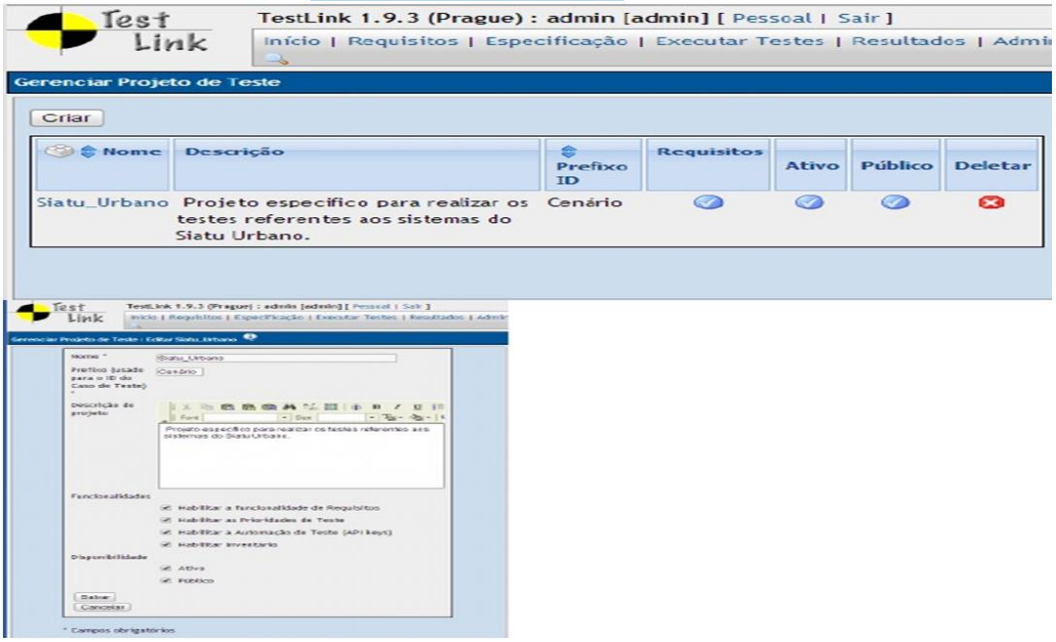
Há ainda o **Test Designer**, que possui permissões para gerenciar a área de Especificação de Teste, criando Suíte de Teste e Casos de Teste, porém não pode executar Casos de Teste.

Projeto de teste	Define um produto de software que será testado
Plano de teste	Lista de casos de teste que serão executados durante um teste
Suíte de teste	Organiza os casos de teste em “pastas”
Caso de teste	Descreve um caso de teste
Build	“Release” ou versão de software

4.1 Telas do TestLink

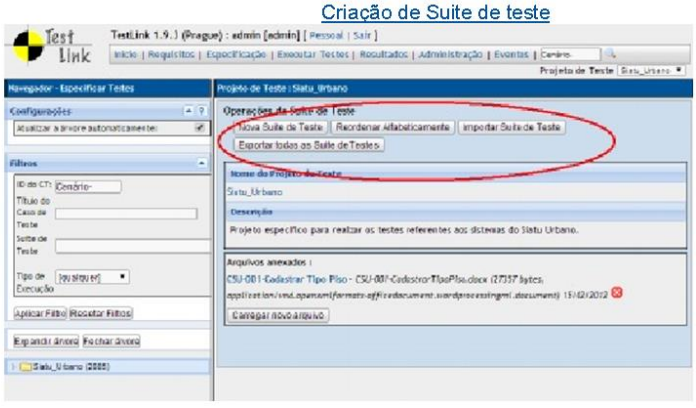
Nessa seção serão apresentadas algumas das principais telas do TestLink:

Gerenciamento de projeto de teste



O administrador pode editar todos os campos referentes ao projeto de teste. Para editar, selecionar o projeto de teste será exibido à janela com todos os dados do projeto habilitados para edição. Edite os dados necessários e clique em Salvar, como mostra a Figura.

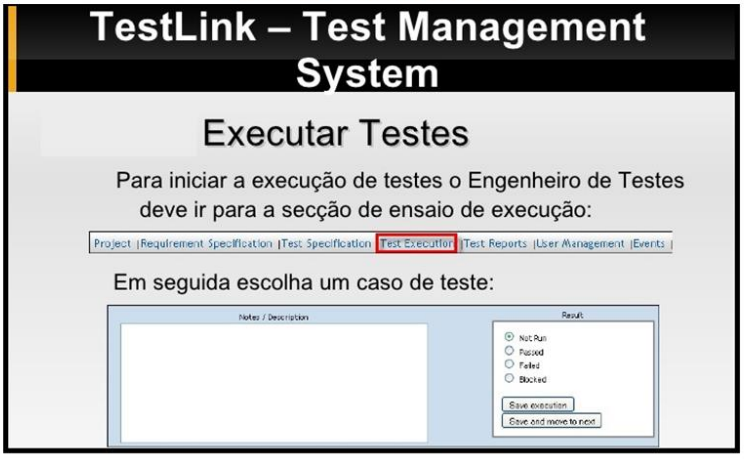
Criação de Suite de teste



A tela abordada na figura acima é usada para criar novos suítes de teste.

Na figura 3.3, inicia-se a execução do teste criado no projeto.

Figura 3. 3– Execução de teste



Na figura 3.4, é mostrada a tela que permite a geração dos relatórios e gráficos que podem ser emitidos pelo TestLink, de modo a facilitar o entendimento dos testes realizados e ressaltas os principais aspectos inerentes aos testes.

Figura 3. 4– Geração de relatório

Conforme foi ilustrado, a ferramenta de teste TestLink é de muito fácil utilização e visualização, por este motivo, entre outros, é que ela foi a escolhida entre tantas opções que existem, para ser apresentada no guia rápido de teste de software.

5. Resultados Finais

A importância de se registrar os resultados dos testes é muito grande: eles são a base para a criação de itens de trabalho para reparo e replanejamento durante o desenvolvimento. E, ao final do desenvolvimento, a compilação dos resultados dos testes bem-sucedidos são um indicativo relevante da confiabilidade daquele software.

Para melhor adequação ao software e a estrutura e porte da empresa deve-se escolher os tipos e técnicas de teste que melhor se adaptem a categoria do sistema e ciclo de vida do mesmo, afim de adequar as constantes evoluções e alterações que venha acontecer.

No mercado competitivo e exigente, a demanda de sistemas ou software cada vez mais complexos com prazo cada vez menores e custos elevados, a atividade de teste de software tornou-se vital para o ciclo de desenvolvimento e sucesso na implantação e utilização.

Após análise do uso prático desse guia, constatou-se a efetiva vantagem da utilização de um guia de rápida visualização das tarefas e para a continuidade deste estudo sugere-se que seja elaborado um manual que defina mais profundamente os testes e crie novos planos de testes especialmente para os profissionais com menos experiência na área e para empresas de menor porte, visto que a economia em termos de tempo e custos foi bem relevante. Abrindo espaço para a elaboração de projetos específicos, para a empresa com o uso de outras ferramentas.

5.1 HABILIDADES DE UM TESTADOR

Um bom profissional de teste é fundamental para o sucesso no processo de teste, por isso serão apresentados as principais competências e habilidades inerentes ao profissional de teste.

- Ser comunicativo, ter facilidade para se comunicar de maneira escrita e oral, com clareza e precisão de modo a transmitir suas ideias e interagir de maneira adequada com a equipe de desenvolvimento.

- Ter visão sistêmica, um bom testador precisa ter uma visão macro, ter a visão do todo, de modo a colaborar com aperfeiçoamento do software que está sendo desenvolvido e entender a sua importância na qualidade do produto final.

- Possuir uma boa visão analítica, deve ter a capacidade de lidar de maneira serena com situações adversas e na analise de problemas. O profissional deve conseguir isolar os problemas para buscar sua causa raiz, trabalhando em cima das soluções e não do problema.

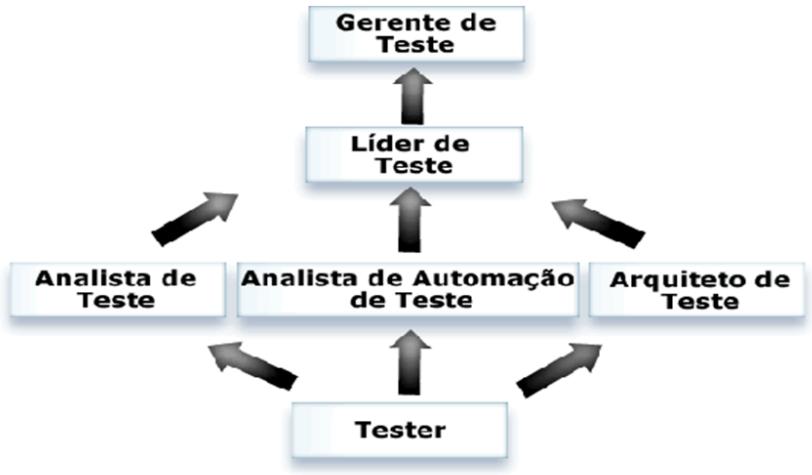
- Pensamento critico, é importante que o profissional não se limite aos testes solicitados nos casos de teste, se aprofunde e busque verificar e validar o maior numero de situações e cenários, em acertas ocasiões pode-se notar que todos os casos de teste da suíte estão passando, e ainda assim, ser gerado um relatório de defeitos sem casos de testes associados a eles . É importante que o profissional sugira melhorias e realize criticas construtivas.

- Pro atividade, é o comportamento de antecipação e de responsabilização pelas próprias escolhas e ações frente às situações impostas pelo meio, é fundamental colaborar com a equipe e buscar sempre se antecipar a possíveis situações adversas.

- Autodesenvolvimento, fazer constantes auto avaliações para identificar pontos fracos ser explorados e melhorados, saber fazer uso da experiência em projetos anteriores.

- Motivação, para sempre estar buscando novos conhecimentos e se adaptar as novas tendências tecnológicas e necessidades.

5.2 EQUIPE IDEAL DE TESTES



A equipe ideal de testes:

- É composta por profissionais cooperativos e com senso de trabalho em equipe
- A equipe como um todo é mais importante que um integrante individualmente
- Todos são responsáveis pelos problemas e pelos resultados obtidos
- Deve possuir profissionais experientes e outros jovens com visões diferentes que se complementem
- Focada nas soluções de problemas
- Sinergia
- Foco nos controles defectivos e nas ações preventivas ante as manutenções corretivas

REFERÊNCIAS

BASTOS, A. et al. **Base de conhecimento em teste e software,** 3 ed. São Paulo: Martins, 2012

COMISSÃO INTERNACIONAL PARA QUALIFICAÇÃO DE TESTE DE SOFTWARE. **Base de Conhecimento para Certificação em Teste.** Disponível em: <https://www.passeidireto.com/ >. Acesso em: 30 jul. 2015.

FAPEG. **Manual do Processo de Teste de Software para Micro e Pequena Empresas versão 2.0**. Goiânia-Goiás, Brasil, 2013.

MOLINARI, Leonardo. Teste de Software. Produzindo Sistemas Melhores e Mais Confiáveis. 1. ed. São Paulo: Érica, 2003.

RIOS, E.; MOREIRA T. **Teste de software.** 3 eds. Revisada e completada. Rio de Janeiro: Alta Books, 2013.

RODRIGUES, Regina. **Tutorial de Utilização do Testlink.** Belo Horizonte, Julho 2014

UFC. **Níveis, Tipos e técnicas de Teste**. Fortaleza-Ceará, Brasil, 2013.

ZHU, H.; HALL P. A. V. e MAY, J. H. R. **Software unit test coverage and adequacy,** ACM Computing Surveys, 1997.