Analista de Testes Certificado – Nível Fundamental (foco em Arquitetura de Testes)

Syllabus

Versão 1.0 – Maio de 2012

(este documento foi produzido originalmente em língua portuguesa do Brasil)

Agradecimentos

Este *Syllabus* foi escrito pelos seguintes membros do IBQTS: Paulo Henrique Nannini e Ronaldo de Almeida.

Esta produção em língua portuguesa do Brasil, sua respectiva revisão e manutenção conta com contribuição voluntária dos membros do IBQTS.

Agradecemos a todos por sua contribuição voluntária.

Prefácio

Objetivo do Documento

Este *Syllabus* define o Nível Fundamental da certificação *Analista de Testes Certificado foco em Arquitetura de Testes* do *Instituto Brasileiro de Qualidade em Testes de Software* (IBQTS).

Este *Syllabus* e seus respectivos exames estão disponíveis junto ao IBQTS em língua portuguesa do Brasil. As instituições responsáveis pela organização de treinamentos podem utilizá-lo como base para a elaboração do material de ensino de seus cursos. Os participantes dos cursos podem utilizá-lo (além de subsídios adicionais na literatura especializada) como preparo para o exame de certificação.

Conteúdo do *Syllabus*

O nível fundamental é dirigido às necessidades de todos os profissionais envolvidos na disciplina de Engenharia de Testes de Software que tem como objetivo adquirir fundamentos essenciais e conhecimento prático de técnicas para a atuação com eficiência na carreira de analista de testes com foco em arquitetura de testes.

Escopo

Os conhecimentos básicos apresentados no nível fundamental são igualmente válidos para todas as áreas (tais como sistemas embarcados, sistemas críticos de segurança, sistemas informação clássicos). Isto não significa, no entanto, que enfoques mais adequados para uma ou outra dessas áreas – sempre levando em consideração suas especificidades – não possam ser abordados em algum curso. O *Syllabus* não tem por objetivo apresentar uma Engenharia de Testes específica para determinada área.

O *Syllabus* não estipula qualquer procedimento específico, nem propõe um modelo de processo específico para o planejamento, monitoramento e sequenciamento dos conceitos aprendidos para a aplicação prática. O *Syllabus* também não visa destacar qualquer processo específico da engenharia de testes, ou mesmo da engenharia de software como um todo.

O *Syllabus* define o conhecimento esperado para analistas de testes com foco em arquitetura, sem no entanto definir a interface exata com outras disciplinas e processos da engenharia de software.

Nível de Detalhamento

O nível de detalhamento deste *Syllabus* possibilita uma consistência de cursos e avaliações em âmbito internacional. Para atingir este objetivo, o *Syllabus* inclui:

- Objetivos gerais de aprendizagem
- Conteúdos que descrevem os objetivos de aprendizagem, e
- Referências a literatura adicional (quando necessário)

Objetivos Educacionais / Níveis de Conhecimento Cognitivo

Cada módulo do *Syllabus* possui um nível cognitivo. Um nível mais alto engloba os níveis inferiores. Os objetivos educacionais são formulados com os verbos "conhecer" para o nível N1 e "dominar e utilizar" para o nível N2. Esses verbos são substitutos para os seguintes verbos em cada nível:

- **N1 (conhecer):** saber, enumerar, caracterizar, reconhecer, nomear, refletir.
- **N2 (dominar e utilizar):** analisar, aplicar, executar, justificar, descrever, avaliar, apresentar, conceber ou projetar, desenvolver, completar, explicar, exemplificar, elicitar, formular, identificar, interpretar, deduzir, atribuir ou caracterizar, distinguir, comparar, compreender, propor, resumir.

Todos os termos definidos no glossário devem ser conhecidos (N1), mesmo não estando expressamente mencionados nos objetivos de aprendizagem.

Estrutura do *Syllabus*

O *Syllabus* consiste de 7 capítulos principais. Cada capítulo equivale a uma unidade de ensino (UE). Cada capítulo principal contém um título, e seu respectivo nível de conhecimento cognitivo e é dividido em sub-capítulos. Além disso, o tempo mínimo de ensino necessário para o capítulo também é indicado, seguido de uma relação de termos importantes do capítulo, que são definidos no Glossário .

Exemplo: UE1 Introdução e Fundamentos (N2)
Duração: 4 horas
Termos: Testes de Software

Este exemplo mostra que o capítulo 1 contém objetivos de aprendizagem do nível N1, e que 4 horas são previstas para o ensino desse capítulo.

Cada capítulo contém sub-capítulos, cujos títulos também indicam o nível de conhecimento cognitivo de seu conteúdo. Os objetivos educacionais (OE) são listados antes do texto do capítulo propriamente dito, mostrando através de sua numeração o sub-capítulo ao qual pertencem.

Exemplo: OE 1.1.1

O número desse exemplo significa que o objetivo educacional OE 1.1.1 está descrito no sub-capítulo 1.1.

O Exame

Este *Syllabus* é a base para o exame de certificação para o *Analista de Testes Certificado – Nível Fundamental (ATC-NF).*

Uma questão do exame pode abranger material de diversos capítulos do *Syllabus*. Todos os capítulos (UE 1 a UE 7) podem ser examinados.

O Exame é presencial, contendo 40 questões no formato de múltipla escolha, com apenas uma única alternativa correta. Todas as questões tem o mesmo peso. Devem ser obtidos pelo menos 70% dos pontos disponíveis.

Histórico de Revisões

Versão	Data	Comentário
1.0	04 de Maio de 2012	Versão Inicial

Política de Versionamento

O versionamento deste documento é determinado por um número composto por dois dígitos.

1. O primeiro dígito mostra a versão do documento dentro da sua evolução. Esse número só é incrementado quando o documento sofre grandes alterações ou inclusão de novos itens.
2. O segundo dígito mostra a versão de correção. Sempre que houver melhoramento de itens já definidos ou correções, este dígito será incrementado.

Política de Revisões

As revisões deste documento são realizados pelos membros do IBQTS, na forma de contribuição voluntária, cedendo automaticamente os direitos autorais para o IBQTS. Todo certificado está apto a ser um membro voluntário, podendo contribuir para a evolução do *Syllabus*, Glossário e Bibliografia, exigidos para as certificações, tanto para o nível fundamental como para os níveis avançado e especialista, tendo citação do seu nome como voluntário nestes materiais. As reuniões de revisões são agendadas e membros voluntários são convocados pelo instituto conforme necessidades observadas.

Sumário

UE 1 Introdução e Fundamentos (N2)

Duração: 4 horas
Termos: Testes de software

Objetivos Educacionais:
OE 1.1.1 Conhecer perfil do profissional de testes
OE 1.2.1 Conhecer as necessidades de conhecimentos, habilidades e atitudes
OE 1.3.1 Conhecer as responsabilidades do analista de testes
OE 1.4.1 Conhecer o histórico dos testes de software
OE 1.5.1 Conhecer a disciplina de testes de software
OE 1.6.1 Conhecer os princípios e conceitos fundamentais de testes de software
OE 1.7.1 Conhecer os momentos de envolvimento dos testes de software
OE 1.8.1 Conhecer o futuro dos testes de software

UE 1.1 Perfil do Profissional de Testes (N2)

Testes de Software é uma atividade intelectualmente demandante, criativa e exploratória, que requer "*insight*", posicionamento, coragem e profunda expertise de comunicação, análise negocial e sistêmica. Para tanto, é necessário definir um perfil básico para o profissional de testes de software. Este profissional deve ter o seguinte perfil:

- realista
- focado a evitar problemas
- analisador
- questionar a criação
- procurar defeitos
- medir a qualidade do produto
- detentor da visão do negócio

UE 1.2 Conhecimento, Habilidades e Atitudes (N2)

O conhecimento caracteriza-se pelo saber, aprimorar, tornar claro aquilo que ainda não se conhece, ou que deseja conhecer mais afundo. O conhecimento é a base de tudo, e pode ser adquirido de várias formas. O analista de testes deve ter/adquirir os seguintes conhecimentos:

- aplicações e plataformas diversas
- metodologias de desenvolvimento e testes
- técnicas e métodos de testes
- análise combinatória
- métricas e estimativas básicas
- *background* em programação (desejável)

Habilidade consiste em praticar o que se conhece, saber fazer. Todo conhecimento que temos é aperfeiçoado com a habilidade que aumenta com a prática. O analista de testes deve ter as seguintes habilidades:

- boa comunicação verbal e escrita
- facilidade de abstração e pensamento criativo
- análise crítica para encontrar erros

- ministrar treinamentos
- ler e interpretar especificações
- observador, paciente e atento aos detalhes

O analista de testes deve querer fazer, arriscar, se comprometer. O analista de testes deve ter as seguintes atitudes:

- integridade e compromisso com a qualidade
- "criatividade destrutiva"
- acreditar que as falhas existem
- encontrar falhas e problemas
- perseguir falhas e não pessoas
- agregar valor

Para torna-se um profissional de sucesso é necessário (1) ampliar seus conhecimentos, (2) aprimorar suas habilidades, e (3) manter o desejo de tornar-se um profissional melhor dia após dia.

UE 1.3 Responsabilidades (N2)

O Analista de Testes com foco em Arquitetura de Testes, é o profissional responsável pelo levantamento das necessidades relacionadas ao processo de teste, incluindo o ambiente de teste, a arquitetura de solução de teste, detalhando a forma de execução e as condições necessárias, os pré-requisitos e restrições tecnológicas, além das ferramentas de teste, quando necessárias.

As responsabilidades do Analista de Testes incluem:

- fornecer subsídios para o planejamento dos testes
- desenvolver casos, cenários e roteiros de testes
- identificar, definir e gerar dados necessários para testes
- fornecer *feedback* do progresso dos testes, tanto da criação quanto da execução
- executar testes
- reportar falhas encontradas
- efetuar testes de especificações

UE 1.4 Histórico dos Testes de Software (N2)

Podemos documentar o histórico da atividade de testes de software, com as seguintes características:

- 1950 – Depuração. A maior facilidade e agilidade na depuração e execução é apresentada como uma estratégia para verificar a corretitude dos programas;
- 1960 – Distinção entre Depuração e Testes;
- 1970 – Profissionalização. O termo "Engenharia de Software" torna-se popular;
- 1972 – A primeira Conferência formal sobre Testes de Software foi organizada por Bill Hetzel e realizada na University of North Carolina em junho de 1972;
- 1973 – Hetzel publica "*Program Test Methods*", apresentando uma análise experimental de métodos de verificação de programas, pela *The University of North Carolina* em *Chapel Hill;*
- 1979, Glenford J. Myers publica "*The Art of Software Testing*", fornecendo uma visão prática, em vez de discussão teórica dos testes de software. Enfatiza metodologias para a concepção de casos de teste eficaz. Exaustivamente abrange a psicologia,

princípios econômicos, aspectos gerenciais, ferramentas de testes, e inspeções de código;

- 1980 – A Era da Qualidade

O enfoque da qualidade é a maneira de como os produtos de determinada empresa são analisados, tendo assim uma melhoria contínua do produto, e considerando que a qualidade seria não só atender , mas surpreender e encantar o cliente. É a Era da Inspeção. Os produtos são verificados um a um. A inspeção encontra defeitos, mas não produz qualidade. Os atributos do produto são examinados, medidos e testados, a fim de assegurar a sua conformidade. Existe duas opções cabíveis (ação corretiva) quando se encontra algum defeito durante a inspeção e o retrabalho.

- 1990 – Observações sobre os benefícios da automação de teste de software em diferentes tipos de organizações não tem sido sistematicamente abrangidas na literatura;
- 2000 – Agilidade em Testes de Software (referências em "*Extreme Testing*" e "*How to Break Software*").

UE 1.5 Disciplina de Testes de Software (N2)

A qualidade de um software está intimamente relacionada às baixas taxas de defeitos encontrados e ao grau de satisfação do usuário na sua utilização, para atendimento de suas necessidades. Lidar com a qualidade de software exige que se estabeleçam mecanismos de garantia de qualidade que assegurem a eficácia nos processos de desenvolvimento de software de modo a obter produtos confiáveis. A atividade de testes de software é um elemento crítico da garantia de qualidade de software e traz consigo uma grande responsabilidade.

Em *"Software Testing In The Real World",* Edward Kit destaca os seguintes itens como essenciais para um processo de testes maduro:

- A qualidade do processo de teste determina o sucesso do esforço de teste
- prevenir a migração de defeitos entre as fases de desenvolvimento, utilizando testes pró-ativos
- utilizar ferramentas de testes
- escolher um responsável pela melhoria do processo de testes
- treinar e capacitar pessoas em testes de software
- cultivar uma atitude positiva de destruição criativa

São consequências de um processo maduro de testes de software os seguintes benefícios:

- maior confiabilidade dos sistemas devido à redução dos defeitos detectados no software após a implantação;
- diminuição de prejuízos ocasionados por defeitos no software;
- aumento do índice de atendimento aos requisitos do cliente;
- redução do custo de retrabalho nos sistemas;
- melhoria da qualidade dos sistemas entregues;
- melhoria contínua no processo de desenvolvimento de sistemas;
- melhoria da imagem da área de TI junto aos clientes.

UE 1.6 Princípios e Conceitos Fundamentais (N2)

O objetivo do processo de testes de software é efetuar a investigação do software com a finalidade de fornecer informações sobre sua qualidade em relação ao contexto em que ele deve operar. Isto significa:

- executar um programa para descobrir defeitos;
- projetar casos de teste com elevada probabilidade de revelar um erro ainda não descoberto;
- projetar testes que descubram sistematicamente diferentes classes de erros com uma quantidade de tempo, esforço e custo mínimos;
- proporcionar boa indicação da qualidade e da confiabilidade do software.

Enfim, podemos concluir que a atividade de testes de software deve contribuir substancialmente em entregar um software de qualidade no menor tempo e com menores custos, para maior lucro do negócio e satisfação profissional de seus clientes e autores.

Testes de Software é uma disciplina profissional, necessitando de pessoas treinadas e capacitadas, identificando rapidamente os problemas mais críticos, justificando um processo de testes melhor, mais rápido e mais barato.

Podemos elencar 7 princípios básicos que norteiam as atividades de testes de software. Estes princípios correspondem a boas práticas de testes que serão abordadas de agora em diante a partir deste momento:

- Desenvolva uma cultura em “encontrar defeitos”: esta é a essência da atividade de testes.
- Pratique testes baseados em riscos: defina uma estratégia de testes para cumprir os prazos, custos e esforços disponibilizados. Testar tudo é caro demais, ou exige um prazo maior do que o prazo fornecido, ou exige esforço maior do que o alocado. Negocie testes.
- Antecipe as atividades de testes: a atividade de testes deve começar o mais breve possível no ciclo de desenvolvimento do software. O planejamento e a preparação dos testes devem ser desenvolvidos antes do software estar pronto.
- Um número pequeno de módulos contém a maioria dos defeitos descobertos durante o teste antes de sua entrega ou exibe a maioria das falhas funcionais.
- Revise e atualize os casos de testes. Pode ocorrer de um mesmo conjunto de testes que são repetidos várias vezes não encontrarem novos defeitos após um determinado momento. Um conjunto de testes novo e diferente precisa ser escrito para exercitar diferentes partes do software ou sistema com objetivo de aumentar a possibilidade de encontrar mais erros.
- Pratique testes baseados em contextos. Testes são realizados de forma diferente conforme o contexto. Software de segurança crítica (por exemplo, software do computador de bordo de aeronaves) são testados diferentemente de um software de comércio eletrônico, e diferentemente um software baseado na web.
- Não deixe a ilusão da ausência de erros contaminá-lo. Encontrar e consertar defeitos não ajuda se o sistema construído não atende às expectativas e necessidades dos usuários.

Existem no mercado inúmeros tipos de teste, com inúmeras nomenclaturas diferentes, muitas vezes o mesmo teste tem nomes diferentes de empresa para empresa.

Independente da metodologia utilizada ou modelo adotado, o teste deve acompanhar todo o ciclo de vida do software, prevenindo a migração de defeitos entre as fases de desenvolvimento, utilizando testes pró-ativos. Somente a qualidade do processo de teste determina o sucesso do esforço de teste.

Testes de Software

Testes do Latim "*testum"* que significa "por a prova" é um processo de verificação e validação de um artefato de software, sendo este uma representação da realidade.

A definição pelo IEEE: Teste (1) Processo que contém as atividades (estáticas e dinâmicas) do ciclo de vida. Trata do planejamento, preparação e avaliação do sistema, e produtos relacionados ao trabalho para verificar se essas atividades satisfazem aos requisitos especificados, para demonstrar que estão de acordo com o objetivo e para detectar defeitos. (2) Conjunto de um ou mais casos de testes.

Algumas outras definições:

- Exame crítico ou prova das qualidades, natureza ou comportamento de uma pessoa ou coisa, *Michaelis*
- O teste consiste em executar o programa com a intenção de encontrar erros, *Glenford G. Myers, 1979*
- O teste não prova nada, mas deve reduzir o risco percebido de não trabalhar dentro de normas e valores aceitáveis, *Boris Beizer.*

Utilizamos comumente a seguinte definição: Testar não é uma atividade, mas uma disciplina mental, que gera produtos de baixo risco, sem esforço significativo.

Focos de Testes de Software

Os focos de testes estão relacionados à maturidade do processo de desenvolvimento e testes de software, que definem os testes como sendo do tipo proativo, reativo ou preditivo. Os 3 focos de testes de software são:

- *Quality Control*

  Baseado em testes reativos, onde as falhas são encontradas apos a construção do software.
- *Quality Assurance*

  Baseado em testes pró-ativos, onde as falhas são encontradas em artefatos intermediários de software, evitando a migração destas falhas para fases seguintes.
- *Quality Design*

  Baseado em Construir a qualidade, estabilidade e testabilidade desde o início da construção do software, com o intuito de evitar falhas e minimizar testes reativos.

Sub-Processo de Testes

O processo de testes de software representa uma estruturação dos sub-processos de testes, que busca a padronização e controle dos testes, buscando maior agilidade. O processo de testes tem o seu próprio ciclo de vida, que é independente do ciclo de vida do desenvolvimento. Comumente utilizamos o acrônimo PPEEG para representar os sub-processos de um processo de testes, onde:

- P: Planejamento de Testes, caracterizado por uma definição de uma proposta de testes baseada nas expectativas do cliente em relação à prazos, custos e qualidade

esperada, possibilitando dimensionar a equipe e estabelecer um esforço de acordo com as necessidades apontadas pelo cliente;
- P: Preparação de Testes, caracterizado pela criação dos casos de testes, agrupados em roteiros de testes, que deverão ser construídos e modificados em função das mudanças solicitadas pelo cliente, bem como pelo próprio aperfeiçoamento do processo de testes (ampliação da cobertura);
- E: Execução de testes, caracterizado pela execução dos roteiros e casos de testes, com o objetivo de garantir que os aplicativos que suportam os processos de negócio estejam "em conformidade" com os requisitos contratados pelos clientes.
- E: Evidenciação de testes, caracterizado pela coleta de evidências da execução dos testes e das falhas encontradas;
- G: Gerenciamento de testes, caracterizado pelo acompanhamento dos testes, gerenciamento de indicadores e criação e manutenção de artefatos.

Fases ou Níveis de Testes

Os níveis de testes correspondem a análise dinâmica ou seja a execução propriamente dita e deve abordar o software de maneiras diferentes, iniciando pelos módulos e prossegue "para fora" na direção da integração do sistema como um todo. É possível realizar testes em 4 níveis diferentes, sendo:

- Unitário: cada unidade é testada individualmente garantindo que ela funcione adequadamente;
- Integrado: as unidades são integradas para formarem um conjunto de funcionalidades de software. Os testes verificam se estas unidades funciona corretamente juntas;
- Sistema: é o teste que visa provar que o software funciona de ponta a ponta, ou seja, da entrada dos dados até o resultado final que pode ser até interagindo com outros sistemas;
- Aceite: é realizado por meio de uma série de testes que demonstram a conformidade com os requisitos acordados com o usuário.

Tipos de Testes

Os tipos de testes são definidos testes funcionais e testes não funcionais. Segue alguns tipos de testes não funcionais:

- Usabilidade
- Portabilidade
- Perfomance
- Carga
- Volume
- Instalação
- Conectivade
- Segurança

Abordagem de Testes de Software

Ao planejar testes deve-se decidir qual ou quais as abordagens serão utilizadas. Para cada momento do ciclo de vida do desenvolvimento de software podemos empregar abordagens

diferentes. Saber combinar diferentes abordagens da maneira certa pode fazer a diferença entre um teste efetuado com sucesso ou não.

São consideradas como abordagens de testes:

- Testes Progressivos

Os testes progressivos são elaborados de acordo com a criação ou a evolução do software. Na abordagem dos testes progressivos, somente é necessário testar as inovações do software, assumindo que nenhum erro foi introduzido após seu processo de desenvolvimento;

- Testes Regressivos

Seu objetivo é assegurar que as alterações em determinadas funcionalidades (durante uma manutenção ou melhorias implementadas) não afetaram outras partes do software. Toda nova versão do software deve ser submetida a um novo ciclo de testes para detectar eventuais impactos em outras funcionalidades.

Os testes progressivos e regressivos podem ser divididos em:

- Testes Positivos

  Teste que visam verificar se uma ou um conjunto de funcionalidades foram desenvolvidas de acordo com as regras especificadas;

- Testes Negativos

  Testes que visam verificar se o software está preparado para ações contrárias ao que se espera;

- Testes Destrutivos

  Testes que visam verificar se situações inesperadas pelo usuários são devidamente previstas pelo software.

Técnicas de Testes

Técnicas de testes existem tanto para testes estáticos como para testes dinâmicos. A proposta das técnicas de testes é auxiliar testadores a encontrarem defeitos como a maior eficácia e eficiência possível.

1. Técnicas de Testes Estáticos

Técnicas de testes estáticos consistem em análise e revisão estática do código. Técnicas de revisão são definidas pela norma *IEEE 1028-2008, Standard for Software Reviews and Audits.*

A finalidade principal de qualquer uma das técnicas de teste estático é encontrar defeitos, mas elas também têm efeitos secundários, por exemplo: *walkthroughs* podem ser utilizados para a troca de conhecimento, e revisões técnicas para a obtenção de consenso. O tipo da técnica de teste estático escolhida não depende somente do tipo de item de teste, mas também dos riscos envolvidos (quanto maior o risco, uma técnica de teste estático mais formal é recomendada).

2. Técnicas de Testes Dinâmicos

Técnicas de testes dinâmicos são técnicas para identificação de condições de testes e consequentemente casos de testes para serem executados em um item de teste, de forma dinâmica. Testes dinâmicos são classificados em 3 grandes categorias baseado em como os inputs são derivados. Essas categorias são:

- Testes baseado em Estrutura (testes de caixa-branca)

Os testes de caixa branca são conhecidos dessa forma porque são baseados na arquitetura interna do software. Esses testes empregam técnicas que objetivam identificar defeitos nas estruturas internas dos programas através da simulações de situações que exercitem adequadamente todas as estruturas utilizadas na codificação. Para realizar os testes de caixa branca é necessário que o profissional de testes conheça a tecnologia empregada pelo software, bem como possua um adequado conhecimento da arquitetura da solução, ou seja, esse profissional deverá ter acesso a fontes, estruturas dos bancos de dados. São técnicas de testes de caixa branca: Teste de Fluxo de Controle (Teste de Caminho e Testes de Laço) e Teste de Fluxo de dados.

- Testes baseado em Especificações (teste de caixa-preta)

Os testes de caixa preta utilizam técnicas para garantir que os requisitos do sistema são plenamente atendidos pelo software. Não é seu objetivo verificar como ocorrem internamente os processamentos no software, mas se o software produz os resultados esperados. Esta técnica não requerer o conhecimento da tecnologia empregada ou dos conceitos de implementação aplicados internamente no software. O conhecimento requerido do profissional de testes é o conhecimento dos requisitos, para que seja possível avaliar o software através dos resultados gerados. Os testes de caixa preta utilizam as seguintes técnicas: Particionamento por Equivalência, Análise de Valor Limite, Tabela de Decisão, Transição de Estados, Árvore de Classificação, Teste de Cenários e Testes Combinatórios.

- Testes baseado na Experiência

Os testes baseados em experiência tem como base a derivação de casos de testes na experiência do testador com itens de testes similares. De forma alternativa, estas experiências podem ser colecionadas e organizadas em *checklists*. As escolha pelas técnicas de testes baseadas em experiência depende: da pessoa que está executando os testes, dos riscos envolvidos, da familiaridade com o item de teste, e com o tipo de defeito esperado para este determinado item de teste. As técnicas mais conhecidas são: *Error Guessing* e Testes Exploratórios.

As técnicas de testes baseadas tanto em estrutura como em especificações são abordadas na UE 3.

Documentação de Testes

Os artefatos desenvolvidos como produtos das atividades de teste, são usados para capturar e transmitir informações do processo de testes, possuindo diretrizes que os descrevem mais detalhadamente. A norma *IEEE Std 829-1998 (IEEE Standard for Software Test Documentation)* define a documentação e relatórios necessários para a execução de um projeto de teste de software. Dentre os relatórios sugeridos estão o Plano de Teste,

Roteiro de Teste, Caso de Teste, Relatório de falhas e Resultados de Teste. Abaixo segue as definições sugeridas.

- Plano de Teste: define e comunica a intenção do esforço de teste. O principal objetivo é ganhar a aceitação e aprovação dos envolvidos no esforço e prazo de teste planejado. Para isso, o documento deve evitar informações que não serão compreendidas ou que serão consideradas irrelevantes pelos envolvidos. É importante que este documento identifique os itens de teste adequados, reflita uma abordagem adequada e possível para a avaliação, considera a amplitude e o riscos envolvidos e identifica precisamente os recursos necessários.
- Roteiro/scripts de teste: a finalidade do roteiro de teste é fornecer informações para a implementação eficiente e efetiva de um conjunto de testes necessários. Contempla instruções passo a passo que permitem a execução de um teste. Os roteiros de teste podem assumir a forma de instruções de texto documentadas e executadas manualmente ou de instruções que podem ser lidas por determinadas ferramentas para ativar a execução automática do teste. Neste último caso chamamos de *script* de teste.
- Caso de Teste: É a definição de um conjunto específico de *inputs* de teste, condições de execução e resultados esperados, identificados com a finalidade de avaliar um determinado aspecto de um Item de teste.
- Relatório de falhas ou relatório de incidentes: registra e consolida as ocorrências que precisam de algum tipo de investigação. Ele é normalmente utilizado para registrar os defeitos encontrados durante a execução dos testes.
- Resultados de Teste: Um conjunto de informações determinadas pela análise de um ou mais registros de teste, que permitem uma avaliação relativamente detalhada da qualidade dos itens de teste e do status do esforço e prazo de teste. Coleta, organiza e apresenta os resultados e as principais indicadores do teste para permitir uma avaliação objetiva da qualidade do produto, recomendações de melhorias, e uma avaliação do processo de testes.

UE 1.7 Momento de Envolvimento (N2)

O modelo de desenvolvimento de software tradicional mostra-se ineficaz, custoso e coloca a fase de testes no caminho crítico da entrega, não favorecendo a prevenção, a identificação, a eliminação e correção das falhas, existindo uma única fase de testes no final do processo de desenvolvimento.

A evolução de um software entre o começo e o final do desenvolvimento pode ser representado por um modelo que representa todas as atividades envolvidas para a sua execução.

O ciclo de vida do processo de testes contempla diversas atividades que devem ser antecipadamente planejadas, tais como os tipos de testes e a serem executados, as abordagens, as estratégias, as técnicas a serem empregadas, o ambiente a ser utilizado e demais informações relevantes para execução dos testes. Dessa forma podemos executar os testes em menos tempo e com uma maior qualidade. Para isso é importante adotar boas práticas como veremos nos capítulos que seguem

UE 1.8 Futuro dos Testes de Software (N2)

Eficiência, agilidade e competitividade são palavras que, cada vez mais, estão sendo empregadas pelas empresas, que, preocupadas em satisfazer - e até mesmo superar - as exigências de seus clientes, resolveram trilhar o caminho da qualidade.

Hoje, não se discute mais se o cliente tem ou não razão. Ele tem razão e fim de conversa. Mais que isso. Com o avanço da informática e a disseminação da informação, o cliente sabe o que quer e exige que suas necessidades sejam plenamente atendidas. Assim, se antes, apesar dos descasos, ele era considerado a razão maior de qualquer negócio, é inegável que, de posse de uma infra-estrutura que lhe dê atualização diária, sua exclusão representa dizer adeus ao mercado em que se atua. Dessa forma, investir em qualidade tornou-se requisito indispensável para as empresas que querem continuar ativas e produtivas.

Problemas Atuais

- maioria das falhas são encontradas tardiamente no ciclo de vida do desenvolvimento de software
- aproximadamente 65% das falhas encontradas em um projeto de software estão relacionadas à discrepâncias, ambigüidades e falhas em requisitos
- cerca de 80% dos testes são executados manualmente

Visão

- reconhecimento dos testadores com visões globais, generalistas, significativo conhecimento em desenvolvimento, automação de testes, e em engenharia de requisitos
- necessidade de redução do custo dos testes de software
- necessidade de construir a qualidade desde o início

Estratégia Proposta

- participação ativa da equipe de testes no processo de desenvolvimento de software
- validar os produtos intermediários de software
- definir meticulosamente os requisitos da aplicação, incluindo problemas a serem resolvidos, e detalhamento das funcionalidades e restrições.

UE 2 Planejamento de Testes (N1)

Duração: 4 horas
Termos: Plano de Teste, Estratégia de Teste, Estimativa de Teste

Objetivos Educacionais:
OE 2.1.1 Conhecer os objetivos do planejamento de teste
OE 2.1.2 Conhecer as atividades do planejamento de teste
OE 2.2.1 Conhecer as estratégia de teste
OE 2.3.1 Conhecer técnicas para estimativa de teste
OE 2.4.1 Conhecer técnicas para monitoramento do progresso de teste
OE 2.4.2 Conhecer técnicas para controle do processo de teste
OE 2.5.1 Conhecer técnicas para encerramento do processo de teste

UE 2.1 Objetivos do Planejamento de Teste (N1)

Planejar testes significa tentar antecipar expectativas (reduzindo o caminho crítico da entrega), definindo o escopo (tamanho do que deve ser testado), estimando o esforço (prazos e recursos necessários), prevendo critérios de qualidade (negociados com o cliente), avaliando riscos (que possam impedir um melhor desempenho), para que qualquer atividade seja suportada por uma estratégia de testes eficaz e eficiente.

Em outras palavras, um bom planejamento de testes de software deve mostrar a importância devida desta atividade, determinar os focos adequados, determinar critérios para criação dos casos de testes, assim como critérios para preparação dos ambientes de execução de testes, determinar recursos (equipe, ambientes e equipamentos) que são adequados e suficientes para a execução, determinar metas e os momentos de análise dos dados e resultados, padronizar atividades e produtos de testes, determinar dos critérios de encerramento ou finalização da atividade. O planejamento de testes facilita a organização dos testes, previsibilidade de custo/esforço, objetividade dos resultados e gerenciamento do processo como um todo.

Planejamento do teste é uma atividade contínua realizada durante todo o processo do ciclo de vida do software. O retorno *(feedback)* da atividade do teste é utilizado para identificar riscos de mudanças, para que ajustes no planejamento sejam efetuados.

O planejamento de teste pode ser documentado em um plano de teste ou de projeto separado em vários planos de testes para diferentes níveis, assim como teste de aceite ou teste de sistemas.

UE 2.2 Estratégia de Teste (N1)

Uma estratégia de teste descreve a abordagem geral e os objetivos das atividades de teste. Ela inclui os níveis de teste (unidade, integração e sistema) que devem ser abordados e os tipos de teste (funcional, desempenho, carga, stress, etc.) que devem ser executados.

A estratégia define:

- Tipos de teste (por exemplo, de funcionalidade, de performance, de usabilidade);
- As técnicas e ferramentas de teste a serem empregadas;
- Os critérios de conclusão e êxito do teste a serem usado (cobertura de teste baseada em código, cobertura de teste baseada em requisitos, número de defeitos, intervalo entre falhas, etc.). Por exemplo, os critérios podem permitir que o software evolua para o teste de aceitação quando 95% dos casos de teste tiverem sido executados com êxito. Outro critério consiste na cobertura de código. Em um sistema no qual a segurança seja vital, esse critério pode significar que 100% do código deve ser coberto por testes.

A estratégia de testes é a conexão entre seu processo de testes e sua missão. É a descrição em alto-nível das fases de testes, com os seus respectivos testes que serão executados, definindo técnicas de projeto de casos de teste e métodos de teste específicos para cada etapa do processo de engenharia de software.

Uma boa estratégia de testes deve conter inspeções, revisões, *walkthrougs*, *peer-reviews*, que não envolve a execução da aplicação e que pode (e deve) ser aplicada durante todo o processo com o objetivo de detectar defeitos. Este teste é denominado de Teste Estático.

Da mesma forma que o teste estático, temos o teste dinâmico esta envolve a execução do software através de simulações com o objetivo de identificar as falhas.

UE 2.3 Estimativa de Teste (N1)

Estimativa: (sf) Avaliação; cálculo; juízo. Num sentido lato, toma-se o significado de estimar como sendo de um cálculo preliminar, sem precisão. Quanto mais preliminar a estimativa mais ajustes faremos ao longo do projeto.

Podemos calcular o esforço dos testes baseado em requisitos e critérios de aceite, em análise de pontos de testes, em análise de pontos de função, em pontos de casos de uso, contagem do número de fluxos e regras de um determinado caso de uso (média de passos de cada fluxo, contagem do número de fluxos alternativos e exceções), contagem das regras de negócio classificadas (simples, média e complexa). Estas estimativas devem ser ajustadas durante todo o processo de teste.

UE 2.4 Monitoramento e Controle (N1)

O propósito da monitoração do progresso do teste é permitir uma visibilidade sobre as atividades do teste. As informações a serem monitoradas podem ser coletadas manualmente ou automaticamente e serem utilizadas para medir os critérios de saída, como cobertura. Métricas podem ser usadas para avaliar o progresso em relação ao orçamento e cronogramas planejados.

As métricas mais comuns incluem:

- Porcentagem de trabalho na preparação do caso de teste (ou porcentagem de casos de testes devidamente planejados)
- Porcentagem de trabalho na preparação do ambiente
- Execução dos casos de testes (números de casos de teste executados ou não, testes com resultados (positivos e negativos)
- Informações dos defeitos (densidade do defeito, defeitos encontrados e resolvidos, taxas de falha e resultado de retestes)
- Cobertura de requisitos, riscos ou código
- Confiança subjetiva do testador sob o produto
- Datas dos pontos de controle
- Custo do teste, incluindo o custo comparado ao benefício de encontrar o próximo erro ou de executar o próximo teste

O controle do teste descreve qualquer orientação ou ação corretiva tomada como resultado de informações e métricas coletadas e relatadas. As ações podem abranger qualquer atividade de teste e pode afetar qualquer outro software de atividade do ciclo de vida ou tarefa.

Exemplos de controle de teste:

- Tomar decisões baseadas em informações adquiridas na monitoração dos teste
- Priorizar novamente os testes quando riscos são identificados.
- Mudar o cronograma de acordo com disponibilidade do ambiente de teste
- Definir um critério de entrada para se iniciar o reteste de defeitos resolvidos pelo desenvolvedor antes de aceitá-lo em uma *build*.

UE 2.5 Encerramento (N1)

O relatório do encerramento teste é constituído de fatos, dados e informações resumidas sobre o que aconteceu durante o período do teste. Informações e métricas devem dar suporte na tomada de decisão e recomendações sobre futuras ações, tais como avaliação

dos defeitos persistentes, vantagem econômica da continuação dos testes e dos riscos consideráveis apontados além do nível de confiança no software testado.

As seguintes informações e métricas podem ser evidenciadas neste relatório:

- A adequação dos objetivos do teste com o nível do teste
- A adequação da estratégia do teste
- O progresso dos testes
- A eficácia dos testes
- A confiabilidade do software testado

UE 3 Técnicas de Validação de Requisitos (N1)

Duração: 4 horas
Termos: Revisão, Requisitos, Portais da Qualidade

Objetivos Educacionais:
OE 3.1.1 Conhecer a conceituação, a motivação e atitudes para testar requisitos
OE 3.2.1 Conhecer as premissas para a revisão de requisitos
OE 3.3.1 Dominar e utilizar técnicas específicas para a revisão de requisitos
OE 3.4.1 Dominar e utilizar métricas específicas referente a revisão de requisitos

UE 3.1 Conceituação, Motivação e Atitudes (N1)

Mais de 60% das falhas graves detectadas pelos testes de aceitação, ou homologação, podem ser rastreados e associados a falhas de especificação; e mesmo assim são poucas as organizações com processos formais de testes e revisões de requisitos ou de especificações. Este tipo de falha representa parcela significativa dos custos de retrabalho posterior, são responsáveis pelo alongamento dos prazos de entrega e pela insatisfação dos clientes. Testes e Revisões Formais de Especificações, realizados ao longo das fases iniciais do desenvolvimento, podem eliminar a maioria destas falhas e diminuir o retrabalho posterior em 70 a 90%, sem comprometer seus prazos de entrega, sem aumentar o *head-count* e assegurando a satisfação dos clientes. Para uma melhor eficácia na revisão de requisitos 9 atitudes são necessárias a serem desenvolvidas e aplicadas:

- Acreditar que é possível testar uma especificação
- Nomear um revisor
- Limitar o tempo e revisão
- Localizar questionamentos
- Registrar os questionamentos
- Delegar ao autor a correção
- Utilizar critérios
- Informar a qualidade da especificação
- Demonstrar o custo/benefício

UE 3.2 Premissas para a revisão de requisitos (N1)

São premissas essenciais para a aplicação de teste de requisitos a existência de requisitos documentados e identificados. A revisão de requisitos é aplicável para todo tipo requisito. Inicialmente deve ser praticado por todos, até mesmo pelos próprios analistas

(especificadores) pois a utilização de um *checklist* auxilia na forma como escrever requisitos.

UE 3.3 Portais da Qualidade (N1)

Os Portais da Qualidade devem ser aplicados a cada requisito individualmente. A aplicação dos Portais da Qualidade colocam os requisitos à prova, verificando diversos aspectos que podem ser fatores de origem de problemas que migram para as fases seguintes, elevando os custos de correção. Requer um investimento profissional e processual, pois é uma atividade que exige disciplina e concentração. A diferença entre as aplicações de teste de requisitos estará no rigor processual e na forma de atuação do revisor.

Os portais da qualidade verificam os seguintes aspectos dos requisitos:

- Completudade, que caracteriza-se pela verificação se o documento recebido é de fato uma especificação;
- Ambigüidade, que caracteriza-se pela verificação das imprecisões, inconsistências, informalidades, falta de clareza, ambigüidades, incoerências e más interpretações;
- Relevância, que caracteriza-se pela verificação da necessidade e importância de cada requisito;
- Testabilidade, que caracteriza-se pela verificação da facilidade de testes dos requisitos;
- Realizabilidade, que caracteriza-se pela verificação da viabilidade da implementação e pela viabilidade dos testes.

UE 3.4 Métricas de Teste de Requisitos (N1)

É possível calcular a eficácia dos testes de requisitos. Para tanto observe em UE 7.4 Indicadores de Qualidade.

UE 4 Técnicas de Testes (N2)

Duração: 8 horas
Termos: casos de teste, técnicas de caixa-branca, técnicas de caixa-preta

Objetivos Educacionais:
OE 4.1.1 Dominar o conceito de condições e casos de teste
OE 4.2.1 Conhecer o propósito das técnicas de testes
OE 4.3.1 Dominar as técnicas baseada em especificação (ou caixa-preta)
OE 4.4.1 Dominar as técnicas baseadas em estrutura (ou caixa-branca)

UE 4.1 Conceitos (N2)

Durante a análise de teste, a estrutura ou documentação base de teste é analisada para determinar o que testar, ou seja identificar as condições de teste. A condição do teste é definida como um item ou evento que pode ser verificado por um ou mais casos de testes. Estabelecer a rastreabilidade das condições de testes de volta até as especificações e requisitos permitem analisar o impacto quando os requisitos mudam e, a cobertura de requisitos a ser determinada por um conjunto de testes.

Durante a modelagem de teste, os casos de teste e os dados de teste são especificados e criados. Um caso de teste consiste de um conjunto de valores de entrada, pré-condições de execução, resultados esperados e pós- condições de execução, desenvolvidos para cobrir certas condições de teste. A *IEEE STD 829-1998* descreve o conteúdo da especificação da modelagem de teste (contendo as condições de teste) e a especificação de caso de teste.

Resultados esperados devem ser produzidos como parte da especificação de um caso de teste e inclui as saídas, mudança de dados e estados, e qualquer outra consequência do teste. Se o resultado esperado não for definido, um resultado errado, pode ser interpretado como correto. Este casos de teste podem ser desenvolvidos especificando o passo a passo para se obter resultados, intermediários ou finais, dependendo da necessidade, considerando-se por exemplo o conhecimento do testador para a execução do teste.

O procedimento de teste especifica a sequência de ações para a execução do teste. Se os testes são executados por uma ferramenta, a sequência de ações é especificada por um *script* automatizado.

A sequência de execução de teste considera alguns fatores como testes de regressão, priorização, dependências técnicas e lógicas.

UE 4.2 Propósito das Técnicas de Teste (N2)

O propósito das técnicas de teste é identificar os casos de teste.

A escolha da técnica adequada dependerá de uma série de fatores, incluindo o tipo de sistema, padrões, clientes, requisitos contratuais, nível do risco, tipos de riscos, objetivos do teste, documentação disponível, conhecimento dos testadores, tempo, dinheiro, ciclo de desenvolvimento, modelo de caso de uso e uma experiência prévia do tipo de defeitos encontrados.

Algumas técnicas são mais facilmente aplicadas em certas situações e níveis de teste, já outras são aplicáveis a todos os níveis.

Ao criar casos de teste, os testadores geralmente usam uma combinação de técnicas de teste.

Classificar testes como caixa-preta ou caixa-branca é uma diferenciação clássica. Técnicas baseadas em especificação (também chamadas de técnicas caixa-preta) são uma forma de derivar e selecionar os casos de testes baseados na análise da documentação ou no seu comportamento, sem levar em consideração a sua estrutura interna. Técnicas estruturais ou baseadas na estrutura (também chamadas de técnicas caixa-branca) são baseadas na estrutura interna de um componente ou software.

Este *syllabus* considera técnicas baseadas em especificação ou técnicas baseadas em experiência como técnicas caixa-preta e técnicas baseadas em estrutura como técnicas caixa-branca.

UE 4.3 Técnicas de Testes baseadas em Especificação (N2)

São características comuns das técnicas baseadas em especificação (ou testes de caixa-preta):

- Modelos, formais ou informais, são utilizados para especificação de um problema a ser resolvido, o software ou seu componente.
- Os casos de testes podem ser derivados sistematicamente destes modelos.

Podem ser definidas as seguintes técnicas:

- Partição de Equivalência
- Análise de Valor Limite
- Tabela de Decisão
- Tabela de Transição de Estados
- Árvore de Classificação
- Teste de Cenários
- Testes Combinatórios

Partição de Equivalência

Na Partição de Equivalência, as entradas do software são divididas em grupos que tenham um comportamento similar, podendo ser tratados da mesma forma. Partições (ou classes) de equivalência podem ser encontradas em dados válidos e inválidos (por exemplo, valores que deveriam ser rejeitados). Partições podem também ser identificadas para valores de saídas, valores internos e valores relacionados a tempo, (antes e após um evento) e para parâmetros de interface (durante teste de integração). Casos de teste podem ser elaborados para cobrir as partições.

Análise de Valor Limite

O comportamento nos limites de uma partição de equivalência é onde existe maior probabilidade de estar incorreto. Portanto, limites são áreas onde casos de testes estão mais propensos a encontrar defeitos. Os valores limites de uma partição são seu máximo e seu mínimo. Um valor limite para uma partição válida é um valor limite válido. O limite de partição inválida é um valor limite inválido. Casos de teste podem ser projetados para cobrir tanto valores válidos como inválidos. Quando os casos de testes são projetados, um valor em cada limite é escolhido.

Análise do valor limite pode ser aplicada em todos os níveis de teste, e sua capacidade de encontrar defeitos é alta. Esta técnica é muitas vezes considerada uma extensão da partição de equivalência e pode ser aplicada para entradas manuais como, por exemplo, em escalas de tempo ou tabela de limites. Valores limites podem também ser usados para selecionar dados de testes.

Tabela de Decisão

A tabela de decisão é considerada uma boa alternativa para capturar requisitos de sistemas que contém condições lógicas e para documentar o comportamento interno do sistema. Elas podem ser utilizadas para registrar regras de negócio complexas a serem implementadas. A especificação é analisada e as condições e ações do sistema são identificadas. As condições de entrada e ações são declaradas de uma forma que possam ser entendidas, como verdadeiras ou falsas (*booleano*). A tabela de decisão contém as condições que disparam as ações, muitas vezes combinações verdadeiras e falsas para todas as condições de entrada, e ações resultantes para cada combinação de condições. Cada coluna da tabela corresponde a uma regra de negócio que define uma única combinação de condições que resulta na execução de ações associadas com aquela regra. A cobertura padrão comumente usada em uma tabela de decisão é ter no mínimo um teste por coluna cobrindo todas as combinações de condições apresentadas. O grande ganho na utilização da tabela de decisão é que ela cria combinações de condições que geralmente não

foram exercitadas durante os testes. Pode ser aplicada a todas as situações quando a execução do software depende de muitas decisões lógicas.

Teste de Transição de Estados

Um sistema pode exibir respostas diferentes dependendo da sua condição atual ou de estado anterior. Neste caso, o comportamento do sistema pode ser representado como um diagrama de transição de estados. Permite ao testador visualizar o software em termos de estados, transições entre estados, as entradas ou eventos que disparam as mudanças de estado (transição) e as ações que podem resultar daquelas transições.

Os estados do sistema, ou objetos em teste, são isolados, identificáveis e finitos. Uma tabela de estado exibe a relação entre estados e entradas, e pode destacar possíveis transições inválidas.

Os testes podem ser construídos para cobrir uma sequência típica de estados, cobrir todos os estados, exercitar todas as transições, exercitar uma sequência específica de transições ou testar transições inválidas.

Árvore de Classificação

O método de árvore de classificação utiliza um modelo do item de teste onde as entradas (parâmetros) do item de teste em partições completamente disjuntos e sub-partições, onde cada partição é uma condição de teste. Todas as divisórias devem ser representadas em uma árvore de classificação hierárquica que ilustra a relação entre as partições. As divisórias deverão ser derivada para os dados de entrada, válidos e inválidos, dependendo do nível de cobertura de teste necessária.

Teste de Cenários

Um cenário descreve um fluxo de processo de um sistema baseado nas suas possibilidades de utilização. Testes de cenários usam modelos das sequências de interação entre um usuário e um item de teste ou entre um item de teste e outros sistemas com o propósito de identificar fluxos de trabalho dentro do item de teste, onde cada sequência de interação (ou seja, cada cenário) constitui uma condição de teste. Isso deve incluir a identificação do cenário "normal" que é a sequência "típica" de ações que são exigidas pelo item de teste, e cenários "alternativos" que representam os fluxos de trabalho alternativos que podem ser realizados através do item de teste (por exemplo, isso pode incluir cenários abrangendo a utilização realista, uso anormal, condições extremas de uso).

Os casos de testes derivados de casos de uso são muito úteis na descoberta de defeitos no fluxo do processo durante a utilização do sistema. Casos de uso muitas vezes são tratados como cenários, e úteis para construir testes de aceite com a participação do usuário final. Teste de cenários são tipicamente usados para a realização de "teste de ponta a ponta" ou *end-to-end testing*, durante os testes de sistema ou testes de aceite do usuário.

Testes Combinatórios

Técnicas de testes combinatórios são usadas para criar casos de teste que permitam atingir vários níveis de cobertura. As combinações de interesse são definidas em termos de parâmetros e os valores destes parâmetros. Uma importante técnica de testes baseada em combinações é a *Pairwise Testing*. Também conhecida como *"all pairs" testing*, *Pairwise*

Testing, é baseada na observação de que a maioria das falhas é causada pela combinação de apenas dois fatores. Em outras palavras, a causa da maioria dos falhas encontradas quando temos várias condições (ou variáveis) de entrada se deve ao conflito entre apenas duas delas. Esta técnica tem como objetivo otimizar os testes, ou seja, reduzir os casos de testes, mantendo uma boa cobertura. A técnica seleciona um conjunto dentre todas as combinações possíveis. Esse conjunto contém todas as combinações entre cada par de variáveis.

UE 4.4 Técnicas de Testes baseadas em Estrutura (N2)

São características comuns das técnicas baseadas em estrutura:

- Informações sobre como o software é construído é utilizada para derivar os casos de testes. Por exemplo, código e informações detalhadas de modelagem.
- A extensão da cobertura do software pode ser medida pelos casos de testes. Além disto, os casos de testes podem ser derivados sistematicamente para aumentar a cobertura.

Técnicas de testes baseado em estrutura visam derivar os casos de teste tais que:

- garantam que todos os caminhos independentes dentro do módulo tenham sido exercitados pelo menos uma vez;
- exercitem todas as decisões lógicas para valores falsos ou verdadeiros;
- executem todos os laços em suas fronteiras e dentro de seus limites operacionais;
- exercitem as estruturas de dados internas para garantir sua validade.

Podem ser definidos as seguintes técnicas:

- Teste de caminho básico
- Teste de condição
- Teste de fluxo de dados
- Teste de laços

Teste de Caminho Básico

O teste do caminho básico deriva uma medida da complexidade lógica do procedimento. Use essa medida para definir um conjunto básico de caminhos de execução. Estes casos de teste garantem a execução de cada instrução do programa pelo menos uma vez durante a atividade de teste.

A complexidade ciclomática é uma métrica de software que proporciona uma medida quantitativa da complexidade lógica de um programa. O valor obtido para a complexidade ciclomática é igual ao número de caminhos independentes do conjunto básico de um programa. A complexidade ciclomática está baseada na teoria dos grafos. O número de regiões do gráfico de fluxo corresponde à complexidade ciclomática.

Teste de Condição

Teste de Condição põe à prova as condições lógicas contidas em um módulo do programa. Tipos de erros de uma condição: Erro de operador booleano; erro de variável booleana; erro de parênteses booleanos; erro de operador relacional; erro de expressão aritmética.

Estratégias de teste de condição:

- Teste de ramos (para uma condição composta).
- Teste de domínio.
- Teste de ramo e de operadores relacionais.

Teste de Fluxo de Dados

O método de teste de fluxo de dados seleciona caminhos de teste de um programa de acordo com as localizações das definições e usos de variáveis no programa.

Teste de Laços

O teste de laços focaliza a validade das construções de laços. Quatro são os tipos de teste de laços: simples, aninhados, concatenados, não-estruturados.

UE 5 Gerenciamento de Falhas (N2)

Duração: 4 horas
Termos: Falhas, Defeitos

Objetivos Educacionais:
OE 5.1.1 Dominar a conceituação para o gerenciamento de falhas
OE 5.1.2 Dominar as principais definições referente falhas
OE 5.2.3 Dominar as convenções referente falhas
OE 5.3.4 Dominar o ciclo de vida das falhas
OE 5.3.5 Dominar os principais atributos para o gerenciamento de falhas
OE 5.3.6 Dominar as boas práticas para o bom reporte das falhas
OE 5.3.7 Dominar a análise de distribuição das falhas
OE 5.3.8 Dominar as técnicas para monitoramento do processo de testes
OE 5.3.9 Dominar os atributos necessários às ferramentas de apoio

UE 5.1 Conceituação (N2)

A base da evolução da engenharia de software é o aprendizado com as falhas. Este aprendizado deve gerar normas, atitudes e hábitos profissionais para minimizar ou eliminar a (re)ocorrência dos mesmos. Como boa prática, registre formalmente as falhas ocorridas e corrigidas antes e depois da produção. A análise causal de falhas é a melhor fonte de informações para identificar o que melhorar em seu processo de desenvolvimento e testes de software.
O processo de gerenciamento de falhas é caracterizado pelas seguintes atividades:

- centralizar e padronizar informações sobre as falhas
- implementar gestão do ciclo de vida da falha
- acompanhar as resolução das falhas
- efetuar análise causal das falhas
- extrair indicadores de qualidade do produto e do processo de testes

Segue abaixo todos os elementos necessários para caracterizar uma falha.

UE 5.2 Definições (N2)

Uma falha de software é uma condição em um produto que não atende a um requisito de software (como indicado nas especificações de requisitos) ou expectativas do usuário final (o que não pode ser especificado, mas são razoáveis). Em outras palavras, uma falha é um defeito na codificação ou lógica que faz com que um software produzir resultados incorretos ou inesperados.

UE 5.3 Convenções (N2)

As seguintes convenções são adotadas:

- Erro: desvio do desejo ou intenção, produzido pelo ser humano
- Defeito: problema introduzido (em qualquer artefato produzido) originado pelo erro
- Falha: manifestação do problema ocasionando um resultado inesperado

Assim, problemas de software são causados por erros provocados por seres humanos. O erro gera um defeito no software. E esse defeito provoca uma falha quando da sua utilização.

UE 5.4 Ciclo de Vida da Falha (N2)

Um *workflow* das falhas, ou seja, o ciclo de vida de uma falha, descreve os estados da falha dele é criado para ele é fechado. A seguir estão alguns termos comumente utilizados para rastreamento de falhas de software importante para acompanhamento e resolução:

- Nova: quando uma falha é recém-criada, ela tem um estado 'nova'. Algumas pessoas separam este estado 'nova' em dois estados, a saber: 'abertas' antes da falha ser atribuída, e 'atribuída' após a falhar ser atribuída. Pode-se ainda ter um estado 'atribuída' se decidir ter todas as falhas recém-submetidas a serem enviadas para um gerente (com estado 'nova'), e o gerente então realmente atribui a falha. Pode-se configurar o *workflow* da falha de tal forma que o próximo estado permitido para 'nova' é 'atribuída'.
- Aberta: algumas pessoas associam uma falha recém-criada e ainda não atribuída a um estado 'aberta'.
- Atribuída: algumas pessoas associam uma falha recém-criada e atribuída a um estado 'atribuída'.
- Corrigida: uma falha que foi corrigida por um desenvolvedor tem um estado de 'corrigida'. Normalmente, este é um estado antes da falha ser confirmada (pela equipe de testes) que realmente foi resolvida. Se uma falha for confirmada ter sido corrigida, ele deve ter um estado de 'fechada'. Alguns sistemas permitem que você configure que, apenas desenvolvedores podem corrigir uma falha. Também chamada de 'resolvida'.
- Fechada: Se uma falha for confirmada (pela equipe de testes) ter sido corrigida, ela deve ter um estado 'fechada'. Alguns sistemas permitem que você configure que, apenas pessoas da equipe de testes pode fechar uma falha.
- Reaberta: Uma falha 'fechada' pode ser reaberta se reapareceu ou se encontra não ser realmente resolvida.
- Suspensa: um falha pode ser 'suspensa' se for determinado que o defeito não deve ser corrigido imediatamente ou uma correção pode ser adiada. Alguns sistemas permitem que você configure que somente uma determinada pessoa, como um gerente, pode suspender uma falha.
- Em Análise: caso for necessária mais informação para corrigir a falha, pode ser convenientemente definir um estado 'em análise'.

UE 5.5 Atributos das Falhas (N2)

As falhas são normalmente são classificadas por:

- Severidade ou Impacto, que é uma classificação para indicar o grau de impacto negativo sobre a qualidade do software. As terminologias reais, e seu significado,

pode variar dependendo de pessoas, projetos, organizações ou ferramentas de rastreamento de defeitos, mas a seguinte classificação é normalmente aceita: (1) Crítica, o defeito afeta a funcionalidade crítica ou dados críticos. A função ou processo é inutilizável. Existe uma solução alternativa para continuar com a atividade; (2) Alta: O defeito afeta uma funcionalidade importante ou dados importantes. Existe uma solução alternativa, mas não é óbvia e é difícil; (3) Baixa: O defeito afeta a funcionalidade de menor importância ou dados não críticos. Existe uma solução fácil. Severidade também é indicada como: S1 = Crítica, S2 = Alta, S3 = Baixa. A severidade do defeito é uma das causas mais comuns de desentendimentos entre os testadores e desenvolvedores. Uma situação típica é quando um testador classifica a gravidade de defeitos como crítico ou importante, mas o desenvolvedor se recusa a aceitar: Ele acredita que o defeito é de gravidade sem baixa. A severidade deve considerar os aspectos técnicos para ser determinada.

- Visibilidade ou Probabilidade, que indica a probabilidade de um usuário encontrar o defeito. Pode ser indicado em percentagem. A medida de probabilidade diz respeito ao uso de uma característica e não a todo o software. Assim, um defeito em um recurso raramente utilizado pode ter uma alta probabilidade caso o defeito seja facilmente encontrado pelos usuários do recurso. Da mesma forma, um defeito em um recurso amplamente utilizado pode ter uma baixa probabilidade se os usuários raramente detectá-lo. Pode-se classificar em : Alta, encontrado por todos ou quase todos os usuários do recurso; Médio, encontrado em cerca de 50% dos usuários do recurso; Baixo, encontrado por muito poucos usuários ou não do recurso
- Prioridade ou Urgência, indica a importância ou urgência de corrigir um defeito. Apesar de prioridade pode ser inicialmente definida pelo testador, geralmente é finalizada pelo Gerente de Projeto ou Produto. Prioridade pode ser classificada nos seguintes níveis: Urgente, que deve ser corrigido na próxima compilação; Alta, que deve ser corrigido de qualquer uma das construções próxima, mas deve ser incluído na versão; Médio, que pode ser corrigido após a liberação, na próxima versão; e Baixa, que pode ou não pode ser corrigida. A prioridade é também indicada como P1 para urgente, P2 para alta e assim por diante. A prioridade deve considerar os aspectos negociais para ser determinada.
- Sistema/Módulo/Componente, indica o módulo ou componente do software onde o defeito foi detectado. Fornece informações sobre quais módulos ou componente oferecem maior riscos.
- Fase Detectada, que indica qual o momento no ciclo de vida de desenvolvimento que a falha foi manifestada.
- Fase Injetada ou Origem, indica a fase no ciclo de vida de desenvolvimento onde o defeito foi introduzido. A fase de origem é sempre antes no ciclo de vida do desenvolvimento que a fase detectada. A fase origem pode ser conhecida somente após uma adequada análise causal.

UE 5.6 Reporte de Falhas (N2)

Documentos detalhando as falhas no software são conhecidos como relatórios de falhas ou de incidentes. O relatório de falhas é um documento técnico que descreve as falhas encontradas no sistema que está sendo testado, também chamado de SUT (*system under test*).

O relatório de falhas deve ser escrito para aumentar a qualidade do produto, documentando um problema de qualidade do SUT:

- serve de instrumento de comunicação com os desenvolvedores, explicando como reproduzir o problema, tornando eficiente o ciclo de vida das falhas com maior agilidade;
- fornece informações para decisão gerencial em relação ao status do produto;
- subsidia a decisão de entrega baseada no status e severidade das ocorrências.

Dois bons relatórios de falhas de um problema podem ser diferentes no estilo e no conteúdo, mas não na essência. Segue 10 sugestões segundo *Rex Black (Quality Week 2000)*:

- Testes estruturados são fundamentais para um bom reporte, facilitando a coleta de informações sobre a ocorrência
- Verifique se é reproduzível, reporte os passos para a reprodução, ocorrências intermitentes devem ser reportadas, inclusive com a incidência,
- Mude as variáveis do teste até isolar o problema e se esforce em identificar a severidade do problema, facilitando a correção
- Verifique se a ocorrência já foi reportada anteriormente, se existe alguma falha similar evitando reporte duplicado
- Verifique se a ocorrência já aconteceu no mesmo teste em versões anteriores ou em condições similares em outros testes nesta mesma versão
- Cada reporte deve ter uma "manchete" descrevendo o problema e o impacto de forma sucinta. Isto dá uma idéia rápida do problema e facilita a consulta das falhas
- Elimine palavras ou passos desnecessários, detalhes e ações irrelevantes ao problema
- Remova palavras ambíguas, subjetivas e vagas que podem causar interpretação incorreta
- Reporte a ocorrência de forma imparcial, focando o problema, sem atacar, criticar, usar humor ou sarcasmo
- Cada reporte deve ser revisado por um ou mais pares para sugestões de melhoria

Alguns cuidados devem ser tomados no processo de reporte das falhas encontradas:

- Não espere para amanhã. Quanto mais demora no reporte da falhas encontrada, menor a chance do defeito ser corrigido a tempo;
- Sempre registre falhas irreproduzíveis e/ou intermitentes, pois podem ser uma bomba relógio;
- A descrição da falha é a informação mais importantes do reporte. Tenha cuidado no tom, não omita e não exagere;
- Tenha certeza que está diante de um problema antes de reportar uma falha, evitando falsos positivos, que geram desperdício de tempo e recursos, além de causar conflitos.

UE 5.7 Distribuição de Falhas (N2)

A análise da distribuição de falhas ajuda a identificar as fraqueza do produto e configurar ações para reduzir o número de defeitos. Os seguintes dados devem estar disponíveis para um período de tempo escolhido e uma determinada versão escolhida:

- falhas por tipo (desenvolvimento, problema de configuração, conhecimento ...)
- falhas por severidade (crítica, alta, baixa)
- falhas por status (aberto, resolvido, fechado,...)

- falhas por tipo de casos de teste (regressivo, progressivo)
- falhas por ambiente (desenvolvimento, homologação, produção)
- falhas por status e tipo
- falhas por status e severidade

UE 5.8 Monitoramento (N2)

O monitoramento do gerenciamento de falhas é uma atividade fundamental. Esta atividade permite assegurar a correção mais rápida do defeito.

As seguintes métricas deverão estar disponíveis a qualquer momento usando as falhas reportadas nos relatórios:

- O número médio de dias antes da resolução de uma falha. Este indicador permite monitorar falhas que insistem em permanecer. Conhecido como *Mean Time to Repair,* ou *MTTR.*
- O tempo médio em cada estado por uma falha. Este indicador permite monitorar defeitos "preguiçosos" e tendências dos defeitos.
- Número de defeitos abertos versus o número de defeitos resolvidos. Este indicador mostra se a organização tem condições para resolver os defeitos encontrados. Esta visão é importante a fim de evitar gargalos de acúmulo de defeitos não resolvidos
- Número de defeitos reincidentes. Este indicador mostra a quantidade de vezes que o defeito foi encaminhado para a equipe de desenvolvimento para sua correção.

Outras métricas podem ser configuradas de acordo com a necessidade, mas as listadas acima são essenciais.

É importante ser capaz de obter a qualquer momento uma lista detalhada de falhas. As listas de defeitos a seguir devem estar disponíveis com todos os detalhes para cada falha:

- Lista geral de falhas descobertas durante um período de tempo,
- Lista de falhas não fechadas.

UE 5.9 Apoio por Ferramentas (N2)

As ferramentas de gerenciamento de falhas são importantes para o processo de testes, na função de armazenar, analisar e fornecer informações sob forma de gráficos. As ferramentas de gerenciamento de falhas devem possuir as seguintes vantagens:

- consolidar informações em um único repositório
- controlar o ciclo de vida das falhas
- facilitar a extração de informações e indicadores
- melhorar a rastreabilidade das ocorrências

Entretanto, alguns pontos de atenção devem ser considerados para o processo de aquisição destas ferramentas:

- Manutenção/Customização
- Extração de Dados
- Suporte/Treinamentos
- Custo

O administrador da ferramenta de gerenciamento de falhas é responsável pela atualização dos valores possíveis para os campos que foram fixados. Cabe ao usuário da ferramenta de gerenciamento de defeitos de informar ao gestor quando um valor adicional é necessário. O

gestor irá validar o pedido e solicitar ao administrador da ferramenta de gerenciamento de falhas para configurar a mudança, se necessário.

UE 6 Automação de Testes (N1)

Duração: 4 horas
Termos: Automação de testes

Objetivos Educacionais:
OE 6.1.1 Conhecer os fundamentos da automação de testes
OE 6.2.1 Conhecer soluções de testes automatizados
OE 6.3.1 Conhecer arquiteturas de testes automatizados

UE 6.1 Fundamentos da Automação (N1)

A automação das atividades de testes de software visa tornar o processo mais ágil em atividades repetitivas, menos suscetível a erros humanos, mais fácil de reproduzir suas atividades e menos dependente da interpretação humana.

UE 6.2 Soluções de Testes Automatizados (N1)

A automação de testes se caracteriza pela construção de soluções de testes automatizados (código de suporte aos testes fora do código funcional) para facilitar e/ou acelerar o processo de execução dos testes. O teste não precisa ser automatizado do início ao fim. Deve-se utilizar a automação como uma assistência. Uma solução automatizada de testes pode ser criada para:

- Gerar instalação do software
- Criar entrada de dados
- Preparar arquivos de dados
- Executar testes
- Analisar resultados
- Coletar evidências

A seguir, alguns exemplos de soluções automatizadas de testes:

- *Drivers*

  Programas escritos para testar (unitário e integrado), que aceitam dados de casos de testes, passam tais dados para o componente, capturam a resposta, analisam os resultados e verificam ou validam se os dados estão corretos gerando um *status report* de testes, tudo automaticamente. Para cada componente deve existir um *driver* de teste; Deve ser reutilizável e reutilizado em testes futuros; Sofre gestão de configuração de software; O componente só poderá ser liberado para testes funcionais quando 100% dos casos de testes do *driver* correspondente tenha sido executado sem falhas.

- *Stubs*

  Programas que simulam funções e programas externos, ainda não construídos, mas necessários para testes integrados, retornando o mesmo valor ou valores diferentes, incluindo condições de erros.

- Simuladores

Programas que simulam ações e resultados de um sistema que não tenho acesso, mas obrigatório para a execução dos testes.

- Extratores

Programas que selecionam ou criam um conjunto de dados para serem utilizados no processo de testes.

- Executor captura e reprodução

Programa construído com auxílio de ferramentas específicas que necessitam de interação com o software (mouse, teclado, eventos), capturando ações e reproduzindo-as num processo de testes funcionais.

UE 6.3 Arquiteturas de Testes Automatizados (N1)

Para execução de testes de forma automatizada, pode-se utilizar as seguintes arquiteturas de automação de testes:

- *Static Capture Replay*

Grava todos inputs, navegações e transações para depois reexecutar o *script* inteiro, Compara o resultado contra o *baseline* que foi criado em uma execução anterior.

- *Data Driven*

Os *scripts* são construídos orientados a dados, que são armazenados e arquivos externos, sua execução segue o mesmo fluxo.

- *Command Driven*

Além de todas as entradas de dados este método sugere conteúdos do tipo "*keywords*" (comandos) que transformam este arquivo de dados em um verdadeiro roteiro de testes.

- *Model-based Test*

Uso de modelos comportamentais que representam a aplicação. Tanto a lógica do caso de teste quanto dos dados utilizados ficam fora do script, utilizando como entrada diagramas de transição de estados.

UE 7 Indicadores de Testes (N2)

Duração: 4 horas
Termos: Medida, Métricas, Medição, Indicadores

Objetivos Educacionais:
OE 7.1.1 Dominar os termos relacionados
OE 7.2.1 Dominar a conceituação dos indicadores de testes
OE 7.3.1 Dominar os indicadores de progresso dos testes
OE 7.4.1 Dominar os indicadores de qualidade de testes
OE 7.5.1 Dominar os benefícios da utilização dos indicadores de testes

UE 7.1 Definições (N2)

Os termos medida, métrica, medição e indicador, costumam causar dúvidas quanto à correta utilização. Em função disso, relacionamos abaixo as definições de cada um deles.

- Medida – é a variável para a qual o valor é atribuído como resultado de uma medição (ISO/IEC 15939, 2002).
- Métrica – é um atributo (propriedade ou característica) mensurável de uma entidade (produto ou processo). No caso de um projeto, uma métrica pode ser, por exemplo, o seu tamanho, uma vez que pode ser medido (MAIA).
- Medição – é o ato de medir, de determinar uma medida. É o conjunto de operações com o objetivo de determinar o valor de uma medida (ISO/IEC 15939).
- Indicador – é a informação relacionada a uma medida, métrica ou combinação de métricas, que pode ser utilizada para compreender a entidade que está sendo medida. É uma métrica que provê uma visão dos processos de desenvolvimento de software e das atividades para melhoria deste processo (BRAUN, 2007).

UE 7.2 Conceituação (N2)

Métricas podem ser coletadas para indicar tanto o progresso dos testes, como a qualidade avaliada dos produtos e do próprio processo de testes.

As métricas de teste são um padrão de medidas muito útil para a verificação da efetividade e da eficiência de diversas atividades do desenvolvimento de software. São obtidas e interpretadas durante o processo de testes. É importante que elas sejam capturadas e utilizadas corretamente para que possam auxiliar na melhoria do processo de desenvolvimento do software através de informações objetivas e pragmáticas para iniciativas de mudanças do processo.

UE 7.3 Indicadores de Progresso de Testes (N2)

As métricas de progresso auxiliam a identificar o status em que se encontra o projeto e a priorizar atividades de forma a reduzir os riscos de não cumprir os prazos estabelecidos, os custos envolvidos, os esforços alocados e escopo acordado.

O progresso dos testes é determinado através da comparação do andamento atual em relação ao planejado. As seguintes grandezas podem ser monitoradas:

- Tamanho (casos de testes desenhados, casos de testes executados, falhas encontradas,...)
- Esforço (tempo dedicado para desenho de casos de teste, tempo dedicado para execução de casos de teste,...)
- Prazo (prazo dedicado para desenho de casos de teste, prazo dedicado para execução de casos de teste,...)

A Curva "S" é um exemplo de gráfico utilizado para a gestão, bastante útil para uso em um programa de métricas de teste. A representação cumulativa atual pode ser comparada com uma curva teórica, que representa a situação ideal, de forma a identificar facilmente os riscos envolvidos na liberação do software para o cliente. A Curva "S" fornece um *feedback* visual imediato do progresso dos testes. O processo de execução dos testes normalmente inicia lento, aumentando o ritmo à medida que o teste continua, quando maiores defeitos são encontrados e mais funcionalidades são liberadas para teste. No final, o ritmo diminui, pois a maior parte dos defeitos já foi encontrada e os casos de teste de menor prioridade são executados.

UE 7.4 Indicadores de Qualidade (N2)

Uma das métricas mais importantes dos testes diz respeito ao número de defeitos encontrados. As falhas fornecem informações sobre a qualidade do produto, do processo e do projeto de teste. Indicadores ajudam a entender o comportamento e funcionamento dos processos, produtos e serviços, indicando:

- a eficácia do processo de testes
- a qualidade de um produto de software
- a percepção da qualidade pelos clientes

É importante a equipe de testes conhecer, coletar, analisar, armazenar e divulgar as informações referente ao processo de desenvolvimento de software. Algumas informações (por exemplo, número de linhas de código produzidas) associadas aos dados coletados referente ao processo de testes (por exemplo, número de falhas encontradas) contribuem para a análise da evolução da qualidade do software.

Avaliando a Eficácia do Processo de Testes

A eficácia de testes é o indicador que evidencia a qualidade do processo de testes em relação ao número de defeitos encontrados. A eficácia de testes é calculada como o percentual de falhas encontradas pelo processo de garantia de qualidade em relação ao total de falhas reportadas. Algumas informações são relevantes:

- Um bom processo de testes tem que ter uma eficácia de pelo menos de 90%
- Um processo de testes barato tem que encontrar falhas no início do processo
- A tendência do mercado é assegurar a qualidade desde o início

A eficácia de testes pode ser calculada considerando o total de falhas encontradas durante o ciclo de vida de desenvolvimento (falhas internas) em relação ao total de falhas encontradas tanto durante o ciclo de desenvolvimento quanto às falhas encontradas em produção (falhas externas).

A eficácia de testes pode ser calculada tanto para o processo de testes de forma global quanto pode ser calculada entre as fases de testes, considerando o ponto de vista dos diversos testes envolvidos. Chamamos de visões da eficácia de testes. Para calcular as diferentes visões da eficácia de testes é necessário determinar o momento no ciclo de vida desejável para efetuar esta coleta.

A eficácia de testes de requisitos também pode ser calculada da mesma forma, sendo necessário coletar a quantidade de falhas encontradas pela atividade de revisão de requisitos e a quantidade de falhas encontradas em fases posteriores com origem em requisitos.

Avaliando a Confiabilidade do Produto

A densidade de falhas é o indicador que evidencia a confiabilidade do produto de software em relação ao número de defeitos encontrados. A densidade de falhas é calculada como a quantidade ou taxa de falhas encontradas em relação ao tamanho do software construído ou mantido, por exemplo em número de linhas de código ou pontos de função.

A densidade de falhas também pode ser calculada tanto para o produto final entregue e monitorado durante um determinado período de tempo, como pode ser calculada entre as fases de testes. Para calcular as diferentes visões da densidade de falhas é necessário determinar o momento no ciclo de vida desejável para efetuar esta coleta.

Essa métrica pode ser usada como base para estimar defeitos em fases futuras ou novas versões

Avaliação a Percepção de Qualidade

Qualidade é aquilo que deixa o cliente satisfeito. As falhas deixam os clientes insatisfeitos. A satisfação dos clientes está diretamente relacionada ao número de falhas de software localizados em produção. Estas falhas localizadas fornecem informações iniciais sobre a estabilidade do software. O número de falhas descobertas, para ter algum significado para análise, deve ser combinado com outras informações, por exemplo tempo de operação ou visibilidade das funcionalidades do software.

O objetivo dessa métrica é alcançar um nível de defeitos que seja aceitável para o cliente. Quanto maior o numero de falhas encontradas pelo cliente, menor e sua percepção de qualidade, maior é a densidade de falhas (calculada em função de um determinado tamanho) e menor será a eficácia do processo de testes.

UE 7.5 Benefícios (N2)

A essência dos testes de software é encontrar e fornecer informações referente ao processo de desenvolvimento e ao produto em questão. As informações coletas são (ou deveriam ser) utilizadas para a tomada de decisões importantes sobre o produto.

As métricas fornece benefícios:

- formação de base histórica para estimativas de projetos futuros
- objetividade e racionalidade gerencial
- mostrar a qualidade do seu trabalho e a competência
- profissionalismo fomentado.

Anexo A: Glossário

O Glossário de Termos do ATC-NF contém os mais importantes termos usados no *Syllabus* IBQTS Analista de Testes Certificado – Nível Fundamental e na disciplina de Engenharia de Testes de Software. Este Glossário este disponível para *download* no endereço http://ibqts.com.br/conteudo/show/id/56.

Anexo B: Lista de Abreviações

OE	Objetivo Educacionais
UE	Unidade de Ensino
IBQTS	Instituto Brasileiro de Qualidade em Testes de Software
NF	Nível Fundamental
N1	Nível Cognitivo 1
N2	Nível Cognitivo 2
IEEE	*Institute of Electrical and Electronics Engineers*
SUT	*System under test*

Anexo C: Referências

O IBQTS disponibiliza uma relação de livros, artigos e sites que merecem destaque especial. Ainda **não** se trata da "Bibliografia Oficial IBQTS" e sim de uma lista de referências. Você também pode encontrá-la em http://ibqts.com.br/conteudo/show/id/56.

Publicado por:

Instituto Brasileiro de Qualidade em Testes de Software (*IBQTS*)

Informações de contato:

contato@ibqts.com.br
Tel.: +55 (11) 4689-0355 (BR)
Endereço postal: Alameda Mamoré, 535 Cj. 1901 06454-040 Alphaville, Barueri - SP - Brasil
Web: www.ibqts.com.br