Qualidade, Testes e Documentação de Software

Brasília-DF.

Elaboração

Jorge Umberto Scatolin Marques

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

Sumário

Apresentação

Caro aluno

A proposta editorial deste Caderno de Estudos e Pesquisa reúne elementos que se entendem necessários para o desenvolvimento do estudo com segurança e qualidade. Caracteriza-se pela atualidade, dinâmica e pertinência de seu conteúdo, bem como pela interatividade e modernidade de sua estrutura formal, adequadas à metodologia da Educação a Distância – EaD.

Pretende-se, com este material, levá-lo à reflexão e à compreensão da pluralidade dos conhecimentos a serem oferecidos, possibilitando-lhe ampliar conceitos específicos da área e atuar de forma competente e conscienciosa, como convém ao profissional que busca a formação continuada para vencer os desafios que a evolução científico-tecnológica impõe ao mundo contemporâneo.

Elaborou-se a presente publicação com a intenção de torná-la subsídio valioso, de modo a facilitar sua caminhada na trajetória a ser percorrida tanto na vida pessoal quanto na profissional. Utilize-a como instrumento para seu sucesso na carreira.

Conselho Editorial

Organização do Caderno de Estudos e Pesquisa

Para facilitar seu estudo, os conteúdos são organizados em unidades, subdivididas em capítulos, de forma didática, objetiva e coerente. Eles serão abordados por meio de textos básicos, com questões para reflexão, entre outros recursos editoriais que visam tornar sua leitura mais agradável. Ao final, serão indicadas, também, fontes de consulta para aprofundar seus estudos com leituras e pesquisas complementares.

A seguir, apresentamos uma breve descrição dos ícones utilizados na organização dos Cadernos de Estudos e Pesquisa.

Provocação

Textos que buscam instigar o aluno a refletir sobre determinado assunto antes mesmo de iniciar sua leitura ou após algum trecho pertinente para o autor conteudista.

Para refletir

Questões inseridas no decorrer do estudo a fim de que o aluno faça uma pausa e reflita sobre o conteúdo estudado ou temas que o ajudem em seu raciocínio. É importante que ele verifique seus conhecimentos, suas experiências e seus sentimentos. As reflexões são o ponto de partida para a construção de suas conclusões.

Sugestão de estudo complementar

Sugestões de leituras adicionais, filmes e sites para aprofundamento do estudo, discussões em fóruns ou encontros presenciais quando for o caso.

Atenção

Chamadas para alertar detalhes/tópicos importantes que contribuam para a síntese/conclusão do assunto abordado.

Saiba mais

Informações complementares para elucidar a construção das sínteses/conclusões sobre o assunto abordado.

Sintetizando

Trecho que busca resumir informações relevantes do conteúdo, facilitando o entendimento pelo aluno sobre trechos mais complexos.

Para (não) finalizar

Texto integrador, ao final do módulo, que motiva o aluno a continuar a aprendizagem ou estimula ponderações complementares sobre o módulo estudado.

Introdução

Basicamente existem dois dispositivos móveis para cada habitante do planeta. Todos os equipamentos que estão sendo lançados estão, de alguma forma, embarcando seus *softwares*. Empresas estão disputando as atenções dos expectadores, clientes e usuários, a cada instante com streaming de vídeo e áudio. Não é só de estrutura física que a demanda está crescendo. A entrega de *software* está em um ritmo alucinado e a tendência é aumentar.

Empresas como a Netflix e Amazon, buscam a especialização na liberação de software para atender as necessidades de seus produtos e clientes. O core de seus negócios, por exemplo, não é somente disponibilizar um filme ou uma série. O *software* que o usuário manipula deve estar funcionando perfeitamente, atualizado e mais rápido que de seu concorrente.

A qualidade e teste de *software* já não é mais um diferencial. E já passou da fase da ciência. Hoje ela é uma necessidade que define quais *softwares* e empresas que permanecerão no mercado.

Outro ponto que atinge diretamente a qualidade são os testes. Testar exaustivamente, automatizado, será fundamental para entregar um produto rápido e sem defeitos.

Portanto, qualidade e testes de *software* caminharão juntos durante todo o ciclo de vida de um software, desde sua ideia de concepção, até seu pleno funcionamento.

Objetivos

- Introduzir o aluno aos estudos sobre Qualidade e Testes de *Software*.
- Avaliar ferramentas e suas formas de utilização.
- Estimular o aluno a descobrir novas utilidades às ferramentas existentes.
- Mostrar ao aluno as diversas normas e *frameworks* relacionado ao tema.

GARANTIA E QUALIDADE DE *SOFTWARE*

UNIDADE I

CAPÍTULO 1

Garantia da Qualidade

Na maioria dos livros sobre Qualidade de *Software*, os autores começam por explicar aos leitores o que é qualidade. Isso é muito importante pois, além de ser algo subjetivo, existem variáveis e questionamentos.

Veja uma breve descrição feita por PIRSIG (2015, p.184, 185):

> Qualidade... a gente sabe o que é, e, ao mesmo tempo, não sabe. Isso é contraditório. Mas algumas coisas são melhores do que outras, ou seja, têm mais qualidade. Porém, se a gente tenta definir qualidade, isolando-a das coisas que a possuem, então já não há o que falar. Se, no entanto, não se pode definir Qualidade, como sabemos o que ela é, ou como sabemos que ela existe? Se ninguém sabe o que é, então, para todos os efeitos, não existe. Mas acontece que, para todos os efeitos, ela existe. Senão, em que se baseariam as notas? Por que as pessoas pagariam fortunas por algumas coisas, jogando outras no lixo? Naturalmente, algumas coisas são melhores que outras... Mas o que é "ser melhor"? Que diabo é Qualidade?

Um bom exemplo é a comparação entre dois carros. Um é carro de passeio que atinge velocidades altas e o outro é uma caminhonete, com rodas largas, 4x4 e possui grande potência. Quais desses dois são os melhores? Perceba que a pergunta, mesmo que a resposta seja dada de imediato por alguém, sempre vai gerar outras novas perguntas, como: "Melhor para o quê?", "Melhor em potência", "Mais rápido?", "Onde será utilizado? ".

Mas então, é possível explicar o que é qualidade? Podemos definir qualidade de maneira resumida ou com poucas palavras?

Não resta nenhuma dúvida que o controle e a garantia da qualidade são tarefas crucias para as empresas que criam e entregam novos produtos ou fazem manutenção em produtos já fabricados. Antigamente, tal processo era função dos artesões que os construíam. Hoje, com o crescimento das demandas, sejam em qualquer ramo de atividade, a melhor prática desse gerenciamento e controle é determinar que outras pessoas façam essa tarefa. No desenvolvimento de *software* não é diferente.

Devido ao grande aumento de dispositivos que utilizam *software* e a necessidade de mantê-los funcionando, a demanda por liberações cada vez mais frequentes se tornou um desafio para as empresas. Entrega frequente e com qualidade requer mudanças culturais e padronizações para que os prejuízos não atrapalhassem o crescimento e a vida saudável dessas empresas.

Qualidade no desenvolvimento de Software

Resumir o conceito de qualidade é um bom início para associar o tema ao desenvolvimento de software. Qualidade em *software* é basicamente atingir os objetivos do cliente, ou seja, os requisitos funcionais e não funcionais. Por exemplo, terminar no prazo e preço combinado, todas as funcionalidades fazendo o que foi proposto e um tempo de vida longo entre melhorias e atualizações. Entretanto, para que isso aconteça é preciso, como não poderia deixar de ser, percorrer um longo, mas, compensador caminho.

Para facilitar a caminhada, é possível procurar por normas e frameworks que darão uma introdução. Quando se fala em *frameworks*, a ideia não é de padronização, mas sim, de boas práticas, diferentemente das normas. “As normas visam fornecer consistentes, rigorosas, uniformes, e métodos aplicáveis para atividades de desenvolvimento e operação de *software*.

O desenvolvimento de padrões, seja por sociedades profissionais como como o IEEE, grupos internacionais como a Organização Internacional para o Comitê Técnico Conjunto 1 da Padronização/Comissão Eletrotécnica Internacional (ISO / IEC JTC1), grupos da indústria ou organizações de desenvolvimento de *software*, está reconhecendo e promovendo que movimento”. (HORCH, 2003, p. 8).

Segundo Gerra (2009, p. 17), “a Qualidade de *software*, que objetiva garantir que especificações explícitas e necessidades implícitas estejam presentes no produto, por meio da definição e normatização de processos de

desenvolvimento. Apesar de os modelos aplicados na garantia da qualidade de *software* atuarem, em especial, no processo, o principal objetivo é garantir um produto que satisfaça às expectativas do cliente, dentro daquilo que foi acordado inicialmente".

Segundo ISSO/IEC 9126-1 (2003, p. 3) define que "a Especificação e avaliação da qualidade do produto de *software* são fatores chave para garantir qualidade adequada. Isto pode ser alcançado pela definição apropriada das características de qualidade, levando em consideração o uso pretendido do produto de software. É importante que cada característica relevante de qualidade do produto de *software* seja especificada e avaliada utilizando, quando possível, métricas validadas ou amplamente aceitas. "

E demonstra que alguns exemplos de usos do modelo de qualidade definido na NBR ISO/IEC 9126, servem para:

- validar a completitude de uma definição de requisitos;
- identificar requisitos de *software*;
- identificar objetivos de projeto de *software*;
- identificar objetivos para teste de *software*;
- identificar critérios para garantia de qualidade;
- identificar critérios de aceitação para produtos finais de software

É sabido, portanto, que para que sejam definidas as necessidades da qualidade, é preciso analisar o contexto da aplicação. Segundo a ISSO/IEC 9126-1, "a avaliação de produtos de *software* com o objetivo de satisfazer as necessidades de qualidade de *software* é um dos processos no ciclo de vida de desenvolvimento de software. A qualidade do produto de *software* pode ser avaliada medindo-se os atributos internos, os atributos externos ou os atributos de qualidade em uso.

Segundo Sommerville (2011, p. 454), foi na década de 1960 que os problemas com a qualidade de *software* foram descobertos, quando ocorreu o desenvolvimento do primeiro sistema de grandes proporções. Durante o século XX, esses problemas persistiram para os engenheiros de software, tornando os produtos lentos e de difícil utilização, prejudicando a expansão de mercado para os sistemas.

Aos poucos foram criadas a adotadas técnicas de gerenciamento de qualidade de *software* desenvolvidas a partir de análises das técnicas já utilizadas pela indústria manufatureira. O nível geral de qualidade de *software* foi sendo aprimorado e

houve melhorias também relacionado ao surgimento de novas tecnologias que facilitaram os processos de testes.

É preciso considerar três principais fatores para um gerenciamento de qualidade de *software* bem realizado:

1. Para o nível organizacional, o gerenciamento de qualidade estabelece um *framework* de processos e padrões que resultem em *softwares* de alta qualidade. A equipe define processos de desenvolvimento bem como a documentação necessária e assim garante o cumprimento dos requisitos do sistema no projeto e no código.

2. Para o nível do projeto, o gerenciamento de qualidade verifica se os processos planejados foram seguidos garantindo a conformidade com os padrões pré-estipulados.

Qualidade do produto e o ciclo de vida do software

Segundo Gerra (2009, p. 18), "Pode-se definir qualidade de produto de *software* como a conformidade a requisitos funcionais e de desempenho declarados explicitamente, padrões de desenvolvimento devidamente documentados e as características implícitas. Por necessidades explícitas, pode-se entender requisitos do usuário; necessidades implícitas relacionam-se, por exemplo, com a performance de execução do sistema ou até mesmo com o cumprimento do cronograma e o orçamento do desenvolvimento do produto".

Outro ponto definido pela ISSO/IEC 9126-1, são as visões da qualidade quando nos referimos a qualidade interna, externa e de uso. Essas visões, mudam durante o Ciclo de Vida do Software, ou seja, tudo que foi definido como requisito no início, difere dos requisitos em outras fases, como por exemplo, na fase de entrega de um produto intermediário, fase de projeto, ou da visão do desenvolvedor. É recomendável que essas perspectivas sejam definidas a fim de que a qualidade seja gerenciada de modo apropriado em cada estágio do ciclo de vida.

Além dessa divisão, as escalas de medidas para as métricas usadas em requisitos de qualidade podem ser divididas entre as categorias correspondentes para os diferentes graus de satisfação dos requisitos.

- **Requisitos de qualidade do usuário**: para criar um produto atendendo as necessidades dos clientes, normalmente é necessária uma

abordagem iterativa no desenvolvimento do software, com feedback constantes, justamente da perspectiva do usuário.

» **Requisitos de qualidade externa**: refere-se aos requisitos derivados das necessidades de qualidade dos usuários, incluindo usabilidade.

» **Requisitos de qualidade interna**: podem incluir modelos estáticos e dinâmicos, outros documentos e código fonte, podem ser usados para definição de estratégias de desenvolvimento e critérios para validação, padronização e verificação durante o desenvolvimento;

» **Qualidade interna** totalização de características do produto de *software* na visão interna.

» **Qualidade externa** é a qualidade que é estimada ou prevista para o produto final de *software* ou para cada estágio de desenvolvimento, baseada no conhecimento da qualidade interna.

» **Qualidade externa** é a totalização de características do produto de *software* do ponto de vista externo.

» **Qualidade em uso estimada** totalização estimada o para o produto final de *software* em cada estágio de desenvolvimento e baseia-se no conhecimento das qualidades internas e externas.

» **Qualidade em uso** é a visão do usuário da qualidade do produto de software.

O modelo a seguir é proposto pela ISO/IEC 9126-1 que define o modelo de qualidade interna e externa, categorizando os atributos em seis características e suas subdivisões:

Tabela 1. Qualidade externa e interna.

Funcionalidade	Confiabilidade	Usabilidade
» Adequação	» Maturidade	» Inteligibilidade
» Acurácia	» Tolerância a falhas	» Apreensibilidade
» Interoperabilidade	» Recuperabilidade	» Operacionalidade
» Segurança de acesso	» Conformidade relacionada à confiabilidade	» Atratividade
» Conformidade relacionada à funcionalidade		» Conformidade Relacionada à usabilidade
Eficiência	**Manutenibilidade**	**Portabilidade**
» Comportamento em relação ao tempo	» Analisabilidade	» Adaptabilidade
» Utilização de recursos	» Modificabilidade	» Capacidade para ser instalado
» Conformidade relacionada à eficiência	» Estabilidade	» Coexistência
	» Testabilidade	» Capacidade para substituir
	» Conformidade relacionada à manutenibilidade	» Conformidade relacionada à portabilidade

Fonte: ISO/IEC 9126-1, (2003).

Outra subdivisão do Modelo de Qualidade definido pela Norma ISO/IEC 9126-1 é o **Modelo de Qualidade em Uso do produto de software**. Nele a preocupação é conceituar e avaliar o *software* sob o ponto de vista do usuário. "Qualidade em Uso é a visão de qualidade que o usuário tem do *software* e é medida em termos do resultado da utilização do software. É a capacidade que o produto de *software* tem de atender aos anseios e às necessidades dos usuários em seu próprio ambiente de trabalho. A avaliação da Qualidade em Uso do *software* valida a qualidade do produto em cenários e tarefas estabelecidas do usuário. "

Os atributos da qualidade em uso são categorizados em quatro características:

- eficácia;
- produtividade;
- segurança;
- satisfação.

Elementos de Garantia da Qualidade de Software

Segundo Pressman (2011, p. 388), "atualmente existe um conceito de grupos de SQA *System Quality Assurance*) ou SQS (*Software* Quality System), que funciona como um serviço de defesa do cliente, ou seja, um grupo que analisará o *software* do ponto

de vista dos clientes, ou seja, perguntas serão feitas para garantir que a qualidade do *software* seja mantida".

Segundo Rosa (2018), "o SQA é organizado em metas, compromissos, habilidades, atividades, medidas e verificações. E tem como objetivo manter a qualidade das operações de produção e os resultados adequados".

Um SQS é criado por vários objetivos, dentre os principais é criar qualidade no ciclo de vida do software, desde sua ideia até o produto final. Segundo Horch (2003, p. 6), isso significa garantir que o problema ou a necessidade seja clara e precisamente declarado e que os requisitos para a solução sejam adequadamente definidos, expressos e compreendidos. Quase todos os elementos de um SQS são orientados para a validade e satisfação de requisitos. Para que a qualidade seja incorporada em um sistema de *software* desde o início, esses requisitos de *software* devem ser claramente entendidos e documentados".

Segundo Horch (2003, p. 8), os oito elementos de um SQS são os seguintes:

Normas

As normas visam fornecer consistentes, rigorosas, uniformes, e métodos aplicáveis para atividades de desenvolvimento e operação de software. O desenvolvimento de padrões, seja por sociedades profissionais como como o IEEE, grupos internacionais como a ISO/IEC JTC1.

A Organização Internacional de Padronização – ISO e a Comissão Eletrotécnica Internacional – IEC formam o sistema especializado para padronização mundial. Organismos nacionais que são membros da ISO ou IEC participam do desenvolvimento de Normas Internacionais por meio de comitês técnicos estabelecidos pela respectiva organização para lidar com campos específicos da atividade técnica. Os comitês técnicos da ISO e IEC colaboram em áreas de interesse mútuo. Outras organizações internacionais, governamentais e não governamentais, em ligação com a ISO e a IEC, também participam do trabalho. No campo da tecnologia da informação, a ISO e a IEC estabeleceram um comitê técnico conjunto, ISO / IEC JTC 1.

Os documentos de normas do IEEE são desenvolvidos nas Sociedades do IEEE e nos comitês de coordenação de padrões do Conselho de Padrões da IEEE *Standards Association* (IEEE-SA). O IEEE desenvolve seus padrões por meio de um processo de desenvolvimento de consenso, aprovado pelo American National Standards Institute, que reúne voluntários que representam pontos

de vista e interesses variados para alcançar o produto final. Os voluntários não são necessariamente membros do Instituto e atuam sem remuneração. Enquanto o IEEE administra o processo e estabelece regras para promover a equidade no processo de desenvolvimento de consenso, o IEEE não avalia, testa ou verifica independentemente a precisão de qualquer informação contida em seus padrões.

Revisão

As revisões permitem uma visibilidade contínua das atividades de desenvolvimento e instalação de software. Revisões de produtos, também chamadas de revisões técnicas, são exames formais ou informais de produtos e componentes em todo as fases de desenvolvimento do ciclo de vida.

Testes

O teste fornece confiança crescente e, finalmente, uma demonstração que os requisitos de *software* estão sendo atendidos. As atividades de teste incluem planejamento, design, execução e relatório.

Análise de defeitos

A análise de defeitos é a combinação de detecção e correção de defeitos e análise de tendências de defeitos. Detecção e correção de defeitos, juntamente com controle de alterações, são um registro de todas as discrepâncias encontradas em cada *software* componente e a disposição de cada discrepância, talvez na forma de um relatório de problemas de *software* ou solicitação de alteração de software.

Gerenciamento de configurações

Sua intenção é manter o controle do software, durante o desenvolvimento e depois que ele é colocado em uso e as mudanças começam.

Segurança

As atividades de segurança se aplicam aos dados e ao próprio data center físico. Essas atividades têm como objetivo proteger a utilidade do *software* e seu ambiente.

Educação

A educação garante que as pessoas envolvidas no desenvolvimento de *software* e as pessoas que usam o software, uma vez desenvolvido, podem fazer seus trabalhos corretamente.

Gerenciamento de fornecedores

Quando o *software* deve ser adquirido, o comprador deve estar ciente e tomar medidas para obter confiança na qualidade do *software* que está sendo adquirido.

Controle de Qualidade e Garantia de Qualidade

QA (*Quality Assurance*) tem por foco a prevenção de problemas nos processos de desenvolvimento, por atividades planejadas e sistemáticas. Enquanto o QC (*Quality Control*) resume-se em atividades ou técnicas utilizadas para capturar problemas, tratar e manter a qualidade do produto.

Segundo Sommerville (2011, p. 455), "Os termos 'garantia de qualidade' e 'controle de qualidade' são amplamente usados na indústria manufatureira. A garantia de qualidade (QA, do inglês quality assurance) é a definição de processos e padrões que devem conduzir produtos de alta qualidade e a introdução de processos de qualidade na fabricação. O controle de qualidade é a aplicação desses processos de qualidade visando eliminar os produtos que não atingiram o nível de qualidade exigido. Na indústria de software, diferentes empresas e de setores industriais interpretam a garantia de qualidade e controle de qualidade de maneiras diferentes. "

CAPÍTULO 2
Métricas de qualidade

Vimos no capítulo anterior sobre definição de qualidade, controle de qualidade e a garantia da qualidade. Para que tudo isso tenha um propósito ou um sentido, é necessário definir um ponto a atingir, ou seja, um objetivo, uma métrica.

Na qualidade de *software* normalmente é definido se os requisitos são cumpridos, ou seja, se os requisitos funcionais ou não funcionais foram satisfeitos. Portanto, cada aplicação, cada produto terá suas métrica a serem atingidas, ficando algumas delas padronizadas de acordo com a norma ou o *framework* adotado. Por exemplo, a norma ISO / EIC 25020: 2009 aplica um modelo de referência para a qualidade do produto de *software*, decompondo-o em características de qualidade e estas, em subcaracterísticas de qualidade, pois essas podem ser avaliadas por medidas de qualidade de produto Programas; como medidas de qualidade são geradas por funções de exibição, que derivam de elementos de medida de qualidade.

Só relembrando:

- **Requisitos funcionais:** especificam o que o *software* deve fazer: cálculos, detalhes técnicos, manipulação e processamento de dados ou qualquer outra função.
- **Requisitos não funcionais:** especificam como o sistema deve funcionar: incluem itens como recuperação de desastres, portabilidade, privacidade, segurança, suporte e usabilidade.

Medir e testar com eficiência a qualidade do seu *software* é a única maneira de maximizar as chances de lançar *software* de alta qualidade nos ambientes de desenvolvimento acelerado de hoje.

Muitos fatores contribuem para definir a qualidade do *software* e algumas formas de medir e acompanhar cada parte de códigos na produção, atenda aos requisitos. Embora a maioria das equipes de desenvolvimento tendem a enquadrar as caraterísticas da qualidade, baseando-se nos requisitos não funcionais, o mais importante é que, ao final, o *software* faça aquilo que ele foi proposto fazer.

Pressman (2011, p. 391), descreve as ações do SQA para garantir um conjunto de metas, atributos e métricas pragmáticas:

Tabela 2. Metas, atributos e métricas.

Meta	Atributo	Métrica
Qualidade dos requisitos	Ambiguidade Volatilidade Clareza no Modelo Facilidade de atribuição	Número de modificadores ambíguos. Número de mudanças por requisito. Número de modelos UML. Número de requisitos não atribuídos ao projeto.
Qualidade do Projeto	Integridade da arquitetura Complexidade da interface	Existência de um modelo da arquitetura. Número de cliques para chegar a uma função.
Qualidade do Código	Padrões Complexidade Facilidade de manutenção Reusabilidade Documentação	Número de padrões usados. Complexidade ciclomática. Percentagem de componentes reutilizados. Índice de legalidade.
Eficácia do Controle de Qualidade	Eficácia de revisão Eficácia nos testes	Erros encontrados e criticidades. Esforço para corrigir um erro. Origem do Erro.

Fonte: adaptada de Pressman (2011, p. 392).

Modelo CISQ

O modelo de qualidade de *software* CISQ define quatro indicadores importantes de qualidade, do código, de *software* que pode ser consultado pelo link: https://www.it-cisq.org/standards/code-quality-standards/.

Tabela 3. Medição de características padrão CISQ.

Característica da qualidade do *Software*	Unidade de práticas de codificação	Nível de sistema de práticas arquiteturais
Confiabilidade	» Protegendo o estado em ambientes multithread. » Uso seguro de herança e polimorfismo. » Gerenciamento de limites de recursos, código complexo. » Gerenciando recursos alocados, tempos limite.	» Conformidade com o design de várias camadas. » *Software* gerencia integridade e consistência dos dados. » Tratamento de exceção através de transações. » Conformidade da arquitetura de classe.
Eficiência de Desempenho	» Conformidade com as melhores práticas orientadas a objetos. » Conformidade com as melhores práticas de SQL. » Cálculos caros em loops. » Conexões estáticas versus pools de conexões. » Conformidade com as melhores práticas de coleta de lixo.	» Interações apropriadas com recursos caros ou remotos. » Desempenho de acesso a dados e gerenciamento de dados. » Gerenciamento de memória, rede e espaço em disco. » Tratamento centralizado de solicitações do cliente. » Uso de componentes da camada intermediária versus procedimentos/funções de banco de dados.
Segurança	» Uso de credenciais codificadas. » Estouros de buffer. » Inicialização ausente. » Validação inadequada do índice da matriz. » Bloqueio inadequado. » Cadeia de formato não controlada.	» Validação de entrada. » Injeção SQL. » Script entre sites. » Falha ao usar bibliotecas ou estruturas examinadas. » Conformidade segura do projeto da arquitetura.
Manutenção	» Código não estruturado e duplicado. » Alta complexidade ciclomática. » Nível controlado de codificação dinâmica. » Sobre parametrização de métodos. » Codificação embutida de literais. » Tamanho excessivo do componente.	» Lógica de negócios duplicada. » Conformidade com o projeto inicial da arquitetura. » Hierarquia estrita de chamada entre camadas arquiteturais. » Camadas horizontais excessivas. » Fan-in/fan-out excessivo em várias camadas.

Fonte: CISQ (https://www.it-cisq.org/standards/code-quality-standards/).

Em seu artigo, Maayan (2018), cita vários exemplos de métricas e métodos de teste para medir os aspectos importantes da qualidade do *software*.

» Contar o **número de bugs** de alta prioridade encontrados na produção.

» Usar o **teste de carga**, que avalia como o *software* funciona em condições normais de uso (cargas altas e cargas baixas).

» **Teste de estresse** é uma variação importante no teste de carga usado para determinar a capacidade operacional máxima de um aplicativo.

» **Segurança** avaliando quanto tempo leva para corrigir ou corrigir vulnerabilidades de *softwares*.

- Contar o **número de linhas de código** é uma medida simples de manutenção. Um exemplo é a complexidade ciclomática, que conta a quantidade de caminhos linearmente independentes através do código-fonte de um programa.
- A **taxa de entrega** contando o número de versões do software.
- **Número de "estórias"** ou requisitos do usuário enviados ao usuário.
- **Testar a GUI (*Graphical User Interface*)** para garantir que seja simples e não frustrante para os usuários finais. O problema é que o teste da GUI é complexo e consome tempo – existem muitas operações e sequências possíveis da GUI que requerem teste na maioria dos *softwares*.

Segundo Sommerville (2011, p. 465), "a medição de *software* preocupa-se com a derivação de um valor numérico ou o perfil para um atributo de um componente de software, sistema ou processo. Comparando esses valores entre si e com os padrões que se aplicam a toda a organização, é possível tirar conclusões sobre a qualidade do *software* ou avaliar a eficácia dos métodos, das ferramentas e dos processos de software."

Para ser considerada uma métrica, é preciso ser uma característica de um processo de desenvolvimento, de documentação, sistema, ou tudo que esteja envolvido na construção de uma aplicação e que possa ser medido. O termo "pode ser medido" não tem muita importância, na maioria das vezes, quando não for comparado com outra medida, ao final de uma etapa para tomarmos uma decisão. Por exemplo, quantidade de erros apresentado no sistema, quantidade de pessoas arrumando erros etc. Esses valores nos trarão algum fundamento para saber se o *software* é viável ou precisa de mais recurso? Porém, faz mais sentido quando calculamos uma porcentagem da quantidade de erros pela quantidade de pessoas alocadas para corrigi-los. Os gerentes usam métricas de processo para decidir se devem ser feitas alterações no processo; as métricas de previsão são usadas para ajudar a estimar o esforço necessário para fazer as alterações no software. (SOMMERVILLE 2011, p. 467)

Segundo Sommerville (2011, p. 467), as métricas de *software* podem ser:

- **Métricas de controle**
  - As primeiras suportam os processos de gerenciamento.
  - Geralmente associadas com os processos de software.

- Exemplo: são o esforço médio e o tempo necessário para reparar os defeitos relatados.

» **Métricas de previsão**

- Ajudam a prever as características do *software*.
- São associadas com o *software* em si.
- São conhecidas como 'métricas de produto'.
- Exemplo: São exemplos de métricas de previsão: a complexidade ciclomática de um módulo.

Metrics and Models for *Software* Quality Engineering. 2. ed. (KAN, S. H. ADDISON-WESLEY, 2003). Discussão muito abrangente sobre as métricas de software que abrange o processo, o produto e as métricas orientadas a objetos. Ela também inclui algumas informações básicas sobre a matemática necessária para desenvolver e entender modelos baseados em medição de software.

Software Quality Assurance: From Theory to Implementation (GALIN, D. ADDISON-WESLEY, 2004). Um olhar excelente, atualizado sobre os princípios e as práticas de garantia de qualidade de software. Inclui uma discussão sobre normas tais como a ISO 9001.

A Practical Approach for Quality-Driven Inspections'. (DENGER, C.; SHULL, F. IEEE Software, v. 24, n. 2, mar.-abr. 2007.). Disponível em: http://dx.doi.org/10.1109/MS.2007.31.

Misleading Metrics and Unsound Analyses (KITCHENHAM, B.; JEFFREY, R.; CONNAUGHTON, C. IEEE Software, v. 24, n. 2, mar.-abr. 2007.). Excelente artigo dos principais pesquisadores de métricas que aborda as dificuldades de compreender o que as medições realmente significam. Disponível em: http://dx.doi.org/10. 1109/MS.2007.49.

The Case for Quantitative Project Management (CURTIS, B. et al IEEE Software, v. 25, n. 3, mai.- -jun. 2008.). Introdução para uma seção especial na revista que inclui dois outros artigos sobre gerenciamento de projetos quantitativos. Argumenta acerca de mais pesquisas em métricas e medições para melhorar o gerenciamento de projetos de software.

Disponível em: http://dx.doi.org/10.1109/ MS.2008.80.

CAPÍTULO 3
Revisões técnicas e inspeções

Primeiramente e sem entrar especificamente em produção de software, vamos definir revisão: uma revisão é um exame sistemático de um documento por uma ou mais pessoas, com o objetivo principal de encontrar e remover erros no início do ciclo de vida de desenvolvimento de software.

As revisões são usadas para verificar documentos, como requisitos, projetos de sistema, código, planos de teste e casos de teste. Normalmente são feitas manualmente nas revisões de processos e utilizando ferramentas em pontos internos relacionado às tecnologias.

A importância da revisão técnica irá refletir diretamente na produtividade da equipe de desenvolvimento. Aos poucos ela é aprimorada e as escalas de tempo reduzidas, pois, a correção de defeitos, nos estágios iniciais e nos produtos de trabalho, ajudará a garantir que esses produtos sejam claros e os custos e prazos de teste são reduzidos, pois há tempo suficiente gasto durante a fase inicial.

Segundo Sommerville (2011, p. 462), **"**revisões e inspeções são atividades de controle de qualidade que verificam a qualidade dos entregáveis de projeto. Isso envolve examinar o software, sua documentação e os registros do processo para descobrir erros e omissões e verificar se os padrões de qualidade foram seguidos".

Pressman (2011, p. 373), "define as revisões como sendo um filtro para a gestão de qualidade, ou seja, as revisão são aplicadas em várias etapas durante o processo de desenvolvimento de software e sua principal função é revelar erros e defeitos que podem ser eliminados". Um defeito ou falha implicariam em problemas de qualidade que, se descobertos depois do *software* ter sido liberado. Portanto, descobrir precocemente um erro eliminariam custos de diversas naturezas.

O esforço para criar uma cultura de revisão é tão necessária que deve se dedicar um grupo específico para isso. Sommerville (2011, p. 463), resume as tarefas e os membros dessa equipe. "Durante uma revisão, um **grupo de pessoas** examina o *software* e a documentação associada à procura de possíveis problemas e não conformidade com padrões. **A equipe** de revisão informada sobre o nível de qualidade de um sistema ou entregável de projeto toma decisões. **Os gerentes** de projeto podem usar essas avaliações para tomar decisões de planejamento e alocar recursos para o processo de desenvolvimento".

Pressman (2011, p. 376), sugere métricas na formação de um grupo para revisão da qualidade:

- esforço de preparação;
- esforço de Avaliação;
- reformulação;
- tamanho do artefato de software;
- erros secundários encontrados;
- erros graves encontrados.

Quando as etapas de produção do *software* estão avançando, elas produzem documentos que podem ser a base das avaliações de qualidade. Tudo pode ser revisto. Desde as especificações, projetos, código, modelos de processos, planejamento de testes, manuais de usuários, ou seja, tudo o que possa ser verificado se os padrões de qualidade foram devidamente seguidos.

Tipos de Revisão

Segundo Mello et. al. (2001, p. 5), as revisões podem ser classificadas em diferentes tipos, dependendo da técnica adotada. Entretanto, a maioria utiliza:

- **Revisões Formais**

  Revisão formal é realizada diretamente pelo cliente, com o objetivo de fornecer feedback para o desenvolvedor. Seu processo é basicamente o seguinte: quando uma fase do ciclo de desenvolvimento estiver concluída, as chamadas desenvolvedor para uma reunião com sua equipe e apresenta o que foi feito para o cliente. Os comentários dos clientes do material e notas que não está claro, o que não corresponde aos requisitos do sistema, e o que está faltando. Em seguida, os desenvolvedores explicam por que o projeto foi desenvolvido como era e essa discussão continua até que um entendimento comum seja alcançado. Em seguida, os desenvolvedores fazer as alterações acordadas entre eles e o cliente.

- **Revisão Interna**

  O ciclo de revisão interna é o mais antigo método de revisão do projeto. Nele, o desenvolvedor distribui cópias do projeto para muitas pessoas,

escolhidas pelo próprio desenvolvedor. Essas pessoas, então, analisam o projeto e dão feedback para o desenvolvedor.

» **Passo a passo**

O passo a passo é um processo de revisão que incide sobre o consenso e, por isso, elimina problemas. Neste processo há um moderador responsável pela direção de ideias e um repórter responsável pela exposição dos problemas encontrados. A discussão é sobre o relatório produzido pela repórter e o moderador é responsável por manter o foco da reunião de revisão, neste caso, passo a passo. Quando há um impasse, o moderador decide isso.

» **Inspeção**

Semelhante, porém mais rigoroso que o tipo passo a passo. O processo de inspeção clássica começa quando o desenvolvedor apresentar o artefato a ser inspecionado e abordar quaisquer perguntas dos colaboradores. Após a apresentação dos desenvolvedores, eles são removidos do processo e só voltam para a reunião para resolver quaisquer problemas que surgem. Neste momento, os revisores estudam o projeto para detectar erros. Posteriormente, o moderador escolhe um repórter que vai apontar os defeitos encontrados.

O processo de revisão

Embora existam muitas técnicas e metodologias de revisões, algumas até não catalogadas, o processo de revisão, em geral, normalmente é estruturado em três fases, segundo Sommerville (2011, p. 462):

» **Atividades preparatórias**

As atividades preparatórias estão relacionadas com o planejamento e a preparação da revisão. O planejamento de revisão envolve a definição de uma equipe de revisão, a organização de um tempo e de um lugar para sua ocorrência e a distribuição de documentos a serem revistos.

» **Reunião de revisão**

Durante a reunião de revisão, o autor do documento ou do programa a ser revisto deve repassar pelo documento com a equipe de revisão. A revisão em si deve ser relativamente curta. Um membro da equipe deve

presidir a revisão e outro deve registrar formalmente todas as decisões e ações a serem tomadas.

» **Atividades pós-revisão**

Após a reunião de revisão, as questões e os problemas levantados devem ser abordados. Esse processo pode envolver a correção de bugs de software, a refatoração do *software* para que ele esteja em conformidade com os padrões de qualidade, ou a necessidade de uma nova redação dos documentos

Equipe de Revisão

Não é padrão o tamanho ou a formação da equipe de revisão, entretanto, se essa equipe for formada, deverá possuir, pelos menos, de três a quatro pessoas, onde o principal é o Projetista Sênior que tem a responsabilidade pela tomada de decisões técnicas significativas. Os demais podem ser os revisores que contarão com o apoio de gerentes de projetos. Ferramentas também poderão ser utilizadas para dar apoio à equipe, umas vez que muitos de seus membros estão geograficamente distantes.

Segundo Sommerville (2011, p. 464), "em desenvolvimento ágil, o processo de revisão de *software* normalmente é informal. Por exemplo, no Scrum, ocorre uma reunião de revisão após a conclusão de cada iteração do software (uma revisão de Sprint), em que os problemas e as questões de qualidade podem ser discutidos. Abordagens ágeis não costumam ser dirigidas a padrões; desse modo, as questões de conformidade com padrões geralmente não são consideradas". Essa falta de procedimentos formais de qualidade em métodos ágeis, restringem empresas que adotam normas a não metodologias ágeis para o desenvolvimento de software.

Segundo Guerra (2009), segue o papel de cada participante da inspeção:

» **Autor:** é o criador (desenvolvedor) do artefato que será inspecionado.

» **Inspetor:** examina o produto antes e durante a reunião de inspeção de modo a tentar encontrar defeitos.

» **Leitor:** pessoa responsável por apresentar o artefato aos demais participantes do processo de inspeção durante a reunião.

» **Escritor:** tem o papel de registrar as informações sobre cada defeito encontrado durante a reunião, que incluem: a localização do defeito, um

resumo do problema, sua classificação e uma identificação do inspetor que o encontrou.

» **Moderador:** o moderador tem o papel mais crítico no processo de inspeção e por este motivo faz-se necessário um treinamento mais aprofundado do que os outros membros da equipe. Ele é a pessoa que lidera toda a equipe e participa ativamente de todas as etapas.

Revisões gerais e Inspeção de *Software*

Revisões de pares de *software*

Esse é um tipo de revisão conduzido para avaliar o conteúdo técnico e a qualidade do trabalho. É normalmente realizada pelo responsável principal do software. Fornece através de discussões uma visão das deficiências do *software* e de seus benefícios. Para corrigir as falhas, o *software* para por novos testes e validações. Existem diversas revisões que fazem parte da revisão por pares de software. Destacam-se:

1. Revisão de Código: análise do código-fonte do *software* e remoção dos erros encontrados no código.

2. Programação em Pares: é um tipo de revisão de código que envolve duas pessoas em uma mesma estação de trabalho.

3. Inspeção: revisão formal na qual a pessoa segue um conjunto de instruções para facilitar a descoberta de falhas. Vários revisores podem realizá-la.

4. Passo a passo: as falhas do *software* são discutidas através de perguntas, comentários e respostas sobre o *software* para elucidar as conclusões sobre o seu perfeito funcionamento.

A equipe de revisão pode encontrar erros lógicos, anomalias no código ou até mesmo recursos que foram omitidos. Analisando em detalhes os modelos de projeto e o código do programa é possível destacar os problemas reais para que sejam reparados.

Os processos ágeis não preveem a utilização da revisão em pares, pois contam com equipes que revisam os códigos uns dos outros e a prática habitual dos programadores revisarem seus próprios códigos.

Os adeptos de Extreme Programming argumentam que a programação em pares é um substituto eficaz para inspeção, pois é um processo contínuo, onde, duas pessoas olham

cada linha de código e verificam antes de este ser aceito, com isso, leva ao profundo conhecimento de um programa, pois ambos os programadores precisam compreender seu funcionamento em detalhes para continuar o desenvolvimento.

Revisão técnica

É a equipe qualificada para examinar a adequação do *software* às especificações e padrões pré-determinados.

Revisões de gerenciamento de *software*

Realizadas pelos representantes da gerência do projeto, que avaliam o status do trabalho e definem as atividades que devem ser realizadas pelo software. Esta revisão é crucial para a tomada de decisões sobre o software.

Revisões de auditoria de *software*

É a avaliação realizada por pessoas externas ao projeto que analisam especificações, padrões e outros critérios atestando suas conformidades.

Segundo Sommerville (2011, p. 464), "durante uma inspeção, frequentemente se usa um *checklist* de erros comuns de programação para ajudar na busca por bugs. Esse checklist pode basear-se em exemplos de livros ou no conhecimento de defeitos comuns em um domínio de aplicação específico. Diferentes *checklists* são usados para diferentes linguagens de programação, pois cada linguagem tem seus próprios erros característicos".

Tabela 4. Um checklist de inspeção

Classe de defeito	Verificação de inspeção
Defeitos de dados	Todas as variáveis de programa são iniciadas antes que seus valores sejam usados?
	Todas as constantes foram nomeadas?
	O limite superior de vetores deve ser igual ao tamanho do vetor ou ao tamanho –1?
	Se as *strings* de caracteres são usadas, um delimitador é explicitamente atribuído?
	Existe alguma possibilidade de *overflow* de *buffer*?
Defeitos de controle	Para cada instrução condicional, a condição está correta?
	É certo que cada *loop* vai terminar?
	As declarações compostas estão posicionadas corretamente entre colchetes?
	Em declarações *case*, todos os *cases* possíveis são considerados?
	Se um *break* é requerido após cada *case* em declarações *case*, este foi incluído?

Defeitos de entrada/saída	Todas as variáveis de entrada são usadas? Todas as variáveis de saída receberam um valor antes de serem emitidas? Entradas inesperadas podem causar corrupção de dados?
Defeitos de interface	Todas as chamadas de funções e métodos têm o número correto de parâmetros? Os parâmetros formais e reais correspondem? Os parâmetros estão na ordem correta? Se os componentes acessam memória compartilhada, eles têm o mesmo modelo de estrutura de memória compartilhada?
Defeitos de gerenciamento de armazenamento	Se uma estrutura ligada é modificada, todas as ligações foram corretamente retribuídas? Se o armazenamento dinâmico é usado, o espaço foi alocado corretamente? O espaço é explicitamente desalocado após não ser mais necessário?
Defeitos de gerenciamento de exceção	Foram levadas em consideração todas as condições possíveis de erro?

Fonte: Sommerville (2011, p. 464).

Ferramentas de Apoio ao Processo de Inspeção

ICICLE

O ICICLE (*Intelligent Code Inspection Environment in a C Language Environment*) é o primeiro *software* de revisão publicado e visa apoiar o processo tradicional de inspeção de software.

Scrutiny

O *Scrutiny* é uma ferramenta colaborativa online, sendo a primeira a permitir que os membros do time de inspeção se encontrassem dispersos geograficamente, podendo ser usada tanto de forma síncrona como assíncrona.

Assist

Asynchronous/ Synchronous Software Inspection Support Tool foi desenvolvida para prover inspeções individuais e em grupo. Como o nome sugere, permite inspeções síncronas e assíncronas, com reuniões tanto em locais diferentes como no mesmo ambiente.

IBIS

Internet-Based Inspection System é uma ferramenta baseada em WEB com notificações por e-mail que auxilia no processo de inspeção desenvolvido. Permite que as inspeções sejam realizadas entre pessoas.

CAPÍTULO 4

Qualidade incremental e melhoria contínua

Mesmo que o projeto de *software* chegue ao final, sem problemas e sem falhas, é altamente recomendável que metodologias de melhoria contínua sejam aplicadas. Isso é importante até mesmo para manter uma base de conhecimento.

Uma das metodologias mais utilizadas para a melhoria contínua, não somente em gerenciamento de projetos, mas em gerenciamento de processos etc. é o PDCA.

PDCA (Plan-Do-Check-Act) é um método de gerenciamento de quatro etapas usado nos negócios para controle e melhoria contínua do processo.

Planejar: o trabalho a ser realizado por meio de um plano de ação após a identificação, reconhecimento das características e descoberta das causas principais do problema.

A primeira etapa, não só dessa metodologia, mas de tudo que se relaciona a projetos, é o planejamento. O planejamento pode ser dividido em três atividades básicas:

- **Identificação do Problema**

  Normalmente, catalogar, documentar, caracterizar um problema é uma atividade muito importante, pois, ocorrer novamente em outro projeto. Resolver problemas e descobrir as soluções para evitá-las no futuro é o objetivo principal da melhoria de teste.

- **Definir o Objetivo**

  Com o entendimento do problema bem definido, fica fácil determinar quais os objetivos de melhoria e o foco em cada fase.

- **Definição das ações**

  Com o problema identificado e o objetivo definido, é possível determinar as ações de melhorias. Esse trabalho deverá ser minucioso pois, pode haver um impacto grande nas etapas anteriores.

Fazer: Realizar o trabalho planejado de acordo com o plano de ação.

Após as definições dos pontos de melhoria é o momento de efetuar a implementação. Para isso, é essencial fazer um questionário com as seguintes perguntas:

- Quais pontos de melhoria precisam ser implementados?
- Quando terminar este plano?
- Quais etapas devem ser tomadas para alcançar o plano?
- Executar ações de melhoria.

Checar: medir ou avaliar o que foi feito, identificando a diferença entre o realizado e o que foi planejado no plano de ação. Nesta fase, o objetivo é verificar se as ações de melhoria foram implementadas com sucesso, bem como avaliar se alcançou a meta desejada. Podemos crias as seguintes atividades:

- Avalie a eficiência das ações de melhoria de teste.
- Meça a eficácia da solução.
- Analise se pode ser melhorado de alguma forma.

A melhor maneira de realizar a avaliação é usar as métricas. As métricas são essenciais para o gerenciamento bem-sucedido da organização e serão tratadas nos próximos capítulos.

Agir: atuar corretivamente sobre a diferença identificada, caso contrário, haverá a padronização e a conclusão do plano, ou seja, ações corretivas sobre os processos de planejamento, execução e auditoria, eliminação definitiva das causas, revisão das atividades e planejamento. (CAMPOS, 1994, p. 33)

Six Sigma

Segundo Pressman (2011, p. 394), o Six Sigma é o processo de produção de alta e melhor qualidade de saída. Isso pode ser feito em duas fases - identificação e eliminação. A causa dos defeitos é identificada e a eliminação apropriada é feita, o que reduz a variação em processos inteiros.

As características do Six Sigma são as seguintes:

- **Controle estatístico de qualidade**

  O termo é derivado de seis desvios-padrão implicando em um padrão de qualidade extremamente elevado.

» **Abordagem Metódica**

O Six Sigma é uma abordagem sistemática de aplicação em DMAIC e DMADV, que pode ser usada para melhorar a qualidade da produção. DMAIC significa *Design-Measure-Analyse-Melhore-Control* e DMADV significa *Design-Measure-Analyse-Design-Verify*.

» **Abordagem baseada em dados e fatos**

O método estatístico e metódico mostra a base científica da técnica.

» **Foco no projeto e no objetivo**

O processo Six Sigma é implementado para se concentrar nos requisitos e condições.

» **Foco no cliente**

O foco no cliente é fundamental para a abordagem Six Sigma. Os padrões de melhoria e controle de qualidade são baseados em requisitos específicos do cliente.

» **Abordagem do trabalho em equipe à gestão da qualidade**

O processo Six Sigma exige que as organizações se organizem para melhorar a qualidade.

Modelos e Recursos de Qualidade de Software

CMMI

O Capability Maturity Model Integration – CMMI é um procedimento e modelo de desenvolvimento de software. Auxilia na organização e racionalização do processo de desenvolvimento de software. Ele avança e impulsiona o processo de desenvolvimento e reduz as ameaças em *software* e sistema.

Estabelecido pelo Instituto de Engenharia de *Software* da Universidade Carnegie Mellon, foi desenvolvido como uma ferramenta de aprimoramento de processo para o desenvolvimento de software. Agora é gerenciado pelo Instituto CMMI. O CMMI analisa seus processos existentes e classifica suas falhas e pontos fortes.

Estabelece cinco níveis de maturidade hierarquizados que, ao serem percorridos, levam as organizações a um nível de excelência em seus processos de desenvolvimento de software.

Tabela 5. Níveis de maturidade CMM.

Nível de Maturidade	Característica
Inicial	Qualquer organização, por pior que seja, está neste nível. Não existe nenhuma condição para uma organização ocupar este nível.
Repetível	As organizações, neste nível, confrontam seus produtos contra requisitos preestabelecidos.
Definido	Os processos são planejados e executados segundo procedimentos conhecidos e entendidos pelas pessoas envolvidas.
Gerenciado	A administração possui indicadores do processo que permitem um acompanhamento quantitativo do seu desenvolvimento.
Otimizado	A organização possui processos extremamente confiáveis e é capaz de prever falhas e alterar o processo e/ou a tecnologia envolvida para obter melhores resultados.

Fonte: adaptada pelo autor de Paulk (1994).

MPS.BR

Segundo a Softex (http://softex.br/mpsbr/modelos/#mpssw),O modelo MPS para *software* (MPS-SW) tem como base os requisitos de processos definidos nos modelos de melhoria de processo e atende a necessidade de implantar os princípios de engenharia de *software* de forma adequada ao contexto das empresas, estando em conformidade com as principais abordagens internacionais para definição, avaliação e melhoria de processos de software.

O modelo MPS possui três componentes:

1. Modelo de Referência para Melhoria de Processo de *Software* – MR-MPS, que fornece uma visão geral sobre os demais guias que apoiam os processos de avaliação e de aquisição.

2. Método de Avaliação para Melhoria de Processo de *Software* – MA-MPS, cujo propósito é verificar a maturidade da unidade organizacional na execução de seus processos de software.

3. Modelo de Negócio para Melhoria de Processo de *Software* – MN-MPS, um resumo executivo que estabelece as regras de negócio e relacionamentos jurídicos dos envolvidos.

Padrões Internacionais de Processo de Software

» ISO 12207: esta norma estabelece uma estrutura comum para processos de ciclo de vida de software, com terminologia bem definida, que pode ser referenciada pela indústria de software. A estrutura contém processos, atividades e tarefas que serão aplicadas durante a aquisição de um produto de *software* ou serviço, e durante o fornecimento, desenvolvimento, operação, manutenção e descontinuidade dos produtos de software. Disponível em: https://www.abntcatalogo.com.br/norma.aspx?ID=38643.

Padrões Internacionais de Qualidade de Software

» **ISO 15504:** A ISO / IEC 15504-6: 2013 constitui um Modelo de Avaliação de Processo, em conformidade com os requisitos da ISO / IEC 15504-2, para a avaliação da capacidade do processo dos processos do ciclo de vida do sistema. A dimensão do processo deste modelo de avaliação de processo é baseada no modelo de referência de processo contido na ISO / IEC 15288.

ISO / IEC 25010

» **ISO / IEC 25010:** define um modelo de qualidade em uso composto por cinco características (algumas das quais subdivididas em subcaracterísticas) relacionadas ao resultado da interação quando um produto é usado em um contexto de uso específico. Este modelo de sistema é aplicável ao sistema humano-computador completo, incluindo sistemas de computador em uso e produtos de *software* em uso. Disponível em: https://www.iso.org/standard/35733.html.

Norma SQuaRE - ISO / IEC 25000

A Série ISO / IEC 25000, denominada Requisitos e Avaliação de Qualidade de Produto de *Software* (SQuaRE), é uma mais moderna norma de qualidade de software, sendo organizada da seguinte forma:

» 2500n: Divisão de gestão da qualidade.

» 2501n: Divisão de modelo de qualidade.

» 2502n: Divisão de utilização de qualidade.

» 2503n: Divisão de requisitos de qualidade.

» 2504n: Divisão de avaliação de qualidade.

O modelo de qualidade do MEDE-PROS

Este modelo de qualidade foi baseado na norma NBR ISO/IEC 9126-1, que padronizou uma linguagem universal para um modelo de qualidade de produto de *software* e é um exemplo de como aplicar os conceitos definidos nesta norma. Este modelo foi amplamente utilizado nas avaliações de qualidade de produto de software, em diversas ocasiões em que o CTI desenvolveu atividades sob demanda. (GUERRA, 2009, p. 81)

Modelo de qualidade do PNAFM

Este modelo de qualidade foi baseado no modelo de qualidade do MEDE-PROS®, que já foi aplicado a centenas (aproximadamente 430) de avaliações de produtos de software. O método foi desenvolvido pela Divisão de Qualificação em *Software* do CTI, para atender ao Programa Nacional de Apoio à Gestão Administrativa e Fiscal dos Municípios Brasileiros. (GUERRA, 2009, p. 81)

VALIDAÇÃO E TESTES

UNIDADE II

CAPÍTULO 1
Conceitos

O motivo pelo qual os testes são realizados é simples de explicar. Sua única missão é mostrar o que o *software* executa, o que está proposto a fazer sem ocasionar erros. No caso de acontecer esses erros, anomalias ou outros eventos não funcionais, medidas de contenções deverão ser tomadas.

Já Sommerville (2011, p. 455) cita dois objetivos distintos:

- Demonstrar ao desenvolvedor e ao cliente que o software atende a seus requisitos.
- Descobrir situações em que o *software* se comporta de maneira incorreta, indesejável ou de forma diferente das especificações.

Segundo Pressman (2011, p. 402), "teste é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente e por isso, deverá ser criado um conjunto de etapas no qual pode-se colocar técnicas específicas de projeto de cada teste e métodos de teste".

Seja qual for a literatura, genericamente as caraterísticas genéricas:

- revisões técnicas;
- inicia no componente em direção à integração do sistema;
- técnicas específicas para diversas abordagens;
- o teste é feito por desenvolvedores e/ou por grupos independentes;
- depuração alinhada com estratégias;

Nessa etapa também entra a equipe de QA em ação. Ela será responsável por gerenciar o processo de teste de release, gerenciam os testes de *software* antes que este seja liberado para os clientes, são responsáveis por verificar se os testes de sistema proporcionam cobertura dos requisitos e mantêm registros adequados do processo de teste.

Muitas vezes o teste de *software* é também conhecido como verificação e validação (V&V). Pressman (2011, p. 402), "define verificação como conjunto de tarefas que garantem que o *software* implementa corretamente uma função específica e, validação refere-se ao conjunto de tarefas que asseguram que o *software* foi criado e pode ser rastreado segundo os requisitos do cliente".

Embora o contexto dos projetos faz com que as atividades de testes e validação, variem de projeto para projeto, existe conceitos básicos que generalizam esse processo.

» **Teste de Desenvolvimento**

É um dos os primeiros testes que se realiza efetivamente. Existem casos que testes são feitos antes de iniciar a codificação, no caso do Extreme Programming, por exemplo. Entretanto, na maioria dos casos, o desenvolvedor realiza os primeiros testes com os componentes criados, de forma independente. Componentes podem ser classes, métodos ou funções, testados separadamente, facilitando uma futura automatização.

» **Testes de sistema**

No teste de Desenvolvimento foram criados e testados Componentes, separadamente. No teste de sistemas, esses componentes serão integrados para criar um subprograma ou parte de um sistema maior. Esse processo pode ser escalável até que o produto como um todo seja atingido. Componentes do sistema são integrados para criar um sistema completo.

» **Testes de aceitação**

Tendo como teste final de uma etapa, ele visa, como o próprio nome diz, a aceitação do usuário. São criados testes, com dados reais, fornecidos e executados pelo próprio cliente, com a finalizada de revelar erros dos quais os desenvolvedores não conseguiram simular ou problemas de requisitos que não atendem as necessidades do cliente.

Processos-- de testes

Sommerville (2011, p. 28), sugere um fluxograma rumo à finalização do produto:

1. Inicialmente, intercala o desenvolvimento e os testes com dados gerados pelo próprio desenvolvedor.
2. Se é utilizada uma abordagem incremental, cada incremento deve ser testado e desenvolvido.
3. Quando o processo é dirigido a planos, uma equipe independente de testadores trabalha a partir desses planos de teste pré-formulados, que foram desenvolvidos a partir das especificações e do projeto do sistema.
4. É criado e executado um teste alfa, até que o cliente aceite.
5. Assim que estiver pronto, e for um sistema comercializável, inicia um teste alfa, assim que o produto for exposto para uso geral.

Agile

É interessante falarmos de processo de testes em práticas ágeis. Criou-se um mito que a escassez de formalidades caminha nos meios dessas metodologias. É bem verdade que pouco *framework* ou normas são direcionadas especificamente para esse propósito. Entretanto, os evangelistas Agile garantem que qualquer processo de teste cabe na metodologia. É preciso, somente, que a prática seja organizada como nas demais formas de desenvolver *softwares*.

Por exemplo, nas metodologias ágeis, o PO é o grande impulsionador da visão baseada nos testes e é muito importante que ele discuta essa visão com a equipe e com o cliente, durante todas as etapas de desenvolvimento. Depois de terminar a visão para um produto, revise-o no início de cada release para determinar se ele ainda atende às necessidades e atualize conforme necessário.

Nos métodos tradicionais, os clientes nunca experimentaram algo assim antes e normalmente ficam impressionados ao ver partes do *software* em funcionamento tão cedo, no ciclo de vida do lançamento. Se eles gostarem da primeira sessão de validação, é muito interessante convidá-los para a próxima revisão de final de *Sprint* e motivá-los, destacando sua importância no processo. Se o *feedback* for positivo, pergunte aos clientes se eles querem participar periodicamente em uma sequência de Sprint.

Uma das vantagens do teste Ágil é o fato de não ser incremental, ou seja, os testes podem ser aplicados a qualquer momento (Sprint) e não somente após a fase de codificação. A equipe trabalha sempre focada em alcançar a qualidade, o prazo é mais curto e todo esforço é voltado para a liberação e entrega.

DevOps

Aos mais experientes no mundo Agile, o DevOps vem se tornando uma prática que está cooperando com as fases de testes. Como visto em capítulos anteriores, para poder participar de times Agile e DevOps, os membros das equipes devem ter um alto nível de coordenação que trafegará entre as funções, desde a entrega até a qualidade, testes e operações. Por ser frequentemente mencionado como parte do Agile, ele inclui conceitos de desenvolvimento contínuo, preenchendo uma lacuna entre o desenvolvimento e controle.

O DevOps é frequentemente referido como uma extensão do Agile que preenche a lacuna entre o desenvolvimento e o controle de qualidade e operações. O DevOps enfatiza muito as ferramentas de automação e integração contínua que permitem o fornecimento de aplicativos e serviços em alta velocidade.

O fato de o teste ocorrer em cada estágio do modelo do DevOps altera o papel dos testadores e a ideia geral do teste. Portanto, para poder efetivamente realizar atividades de teste, espera-se que os testadores possuam habilidades técnicas e até tenham conhecimento de código.

Característica de um *software* testável

Portanto, mesmo antes de iniciar o primeiro rascunho da aplicação, é necessário estar pronto para os processos de testes para garantir a sua testabilidade. Segundo Pressman (2011, p. 429), as seguintes características levam a um *software* testável:

- **Operabilidade**

  Quanto mais um sistema for projetado para melhor funcionar, mais eficiente será para ser testado.

- **Observabilidade**

  O que você vê é o que você testa. O código fonte é acessível.

» **Controlabilidade**

Quanto mais controle sob o software, melhor será a automatização e otimização dos testes.

» **Decomponibilidade**

Se o *software* for construído com a ideia de módulos independentes, melhor será para testá-los individualmente.

» **Simplicidade**

Quanto menos tiver para testar, mais rapidamente será concluído o teste.

» **Estabilidade**

Quanto menos alteração, menos interrupção nos testes.

» **Compreensibilidade**

Quanto mais informações, em mais inteligente será o teste.

Processos de melhoria de testes

TPI

Segundo (GUERRA, 2009, p. 37), "o modelo Test Process Improvement - TPI foca na melhoria do processo de testes e ajuda a definir gradualmente os passos para sua evolução, levando em consideração o tempo, o custo e a qualidade. Foi desenvolvido porque a IQUIP (Intelligent & Quick Information Processing) e seus clientes necessitavam de um modelo que suportasse as constantes mudanças do processo de testes".

Test Maturity Model - TMM

"O Testing Maturity Model – TMM foi desenvolvido pelo Illinois Instituty of Tecnology como um guia para melhoria de processos de testes. A estrutura do TMM foi baseada no CMM, e está aderente ao CMMI, consistindo em cinco níveis que avaliam o grau de maturidade de um processo de testes". (GUERRA, 2009, p. 39).

Test Improvement Model - TIM

Desenvolvido pela Ericson, Subotic e Ursing o TIM foi concebido pelos desenvolvedores que sentiam a necessidade de melhorar o processo de testes. O TIM se propõe a identificar o estado atual das práticas das áreas chaves e serve como um guia na implementação dos pontos fortes e na remoção dos pontos fracos.

CAPÍTULO 2
Métodos de validação e testes

Segundo Pressman (2011, p. 430), “qualquer produto de engenharia pode ser testado, conhecendo a função específica, para qual foi projetado (caixa-preta) ou conhecendo seu funcionamento interno, ou seja, se seus componentes internos foram adequadamente exercitados (caixa-branca). “

Teste de caixa preta

Esse método recebe esse nome porque a equipe de controle de qualidade se concentra nas entradas e nas saídas esperadas sem saber como o aplicativo funciona internamente e como essas entradas são processadas. O objetivo deste método é verificar a funcionalidade do software, garantindo que ele funcione corretamente e atenda às demandas do usuário.

Segundo Pressman (2011, p. 439), o teste de caixa-preta tenta encontrar as seguintes categorias de erros:

- funções ou métodos incorretos ou faltando;
- erros de interface;
- erros em estrutura de dados;
- erros em acesso à base de dados;
- erro de comportamento;
- erros de inicialização e término.

Após conseguir os objetivos, o próximo passo é verificar se todos os objetos possuem o relacionamento esperado. Para isso, Pressman (2011, p. 440) sugere:

- **Métodos baseados em grafos**: é criado um grafo e as relações entre os objetos.
- **Particionamento de equivalência:** divide o domínio de entrada em programa de classes de dados para serem criados os testes.

- **Análise de valor limite:** analisam erros que ocorrem nas fronteiras do domínio, leva em consideração seleção de casos de teste que utilizam valores limites e se parece com o particionamento de equivalência.
- **Matriz ortogonal:** é utilizado para problemas de domínios pequenos, porém, muito grandes para testes de exaustão. Utilizado para encontrar falhas em regiões.

Um teste de caixa preta importante é o Teste Funcional. Nele, o sistema é testado em relação aos requisitos funcionais, alimentando-o e examinando a saída. Consequentemente, dá significado não ao próprio processamento, mas aos seus resultados. O teste funcional geralmente é realizado dentro dos níveis de sistema e aceitação. Normalmente, o processo de teste funcional compreende o seguinte conjunto de ações:

1. Descreve as funções que o *software* deve executar.
2. Compõe os dados de entrada dependendo das especificações da função.
3. Determina a saída dependendo das especificações da função.
4. Executa o caso de teste.
5. Sobrepõe as saídas recebidas e esperadas.

Particionamento de equivalência

Nesta técnica, todo o intervalo de dados de entrada é dividido em diferentes partições. Todos os casos de teste possíveis são considerados e divididos em um conjunto lógico de dados chamado classes. Um valor de teste é escolhido de cada classe durante a execução do teste.

Teste de Tabela de Decisão

Nessa técnica, os casos de teste são projetados com base nas tabelas de decisão formuladas usando diferentes combinações de entradas e suas saídas correspondentes com base em várias condições e cenários que aderem a diferentes regras de negócios. Veja um bom exemplo de tabela de decisão nesse artigo: https://www.guru99.com/decision-table-testing.html, acesso em: 18/1/2010.

Teste da caixa branca

Ao contrário do teste de caixa preta, esse método requer conhecimento profundo do código, pois envolve o teste de alguma parte estrutural do aplicativo. Portanto, geralmente, os desenvolvedores diretamente envolvidos na escrita de código são responsáveis por esse tipo de teste. Este método é usado principalmente nos níveis de teste de unidade e integração.

Segundo Pressman (2011, p. 431), usando o método de caixa-branca, é possível criar casos de teste que:

- garantam que os caminhos foram exercitados ao menos uma vez;
- exercitam todas as decisões lógicas de seus estados;
- executam todos os ciclos em seus limites;
- exercitam estruturas de dados internas para assegurar sua validade;

Teste baseado em modelos

É um teste de caixa-preta que utiliza modelos comportamentais baseado em UML. Para isso, Pressman (2011, p. 439), sugere:

1. Analise um modelo comportamental existente para o software, ou crie um.
2. Percorra o modelo comportamental e especifique as entradas que forçarão o *software* a fazer transição de um estado para outro.
3. Reveja o modelo comportamental e observe as saídas esperadas à medida que o *software* faz a transição de um estado para outro.
4. Execute os casos de teste.
5. Compare o resultado real e esperado e tome a ação necessária.

Teste de performance

O teste de desempenho visa investigar a capacidade de resposta e a estabilidade do desempenho do sistema sob uma determinada carga. Dependendo da carga de trabalho, um comportamento do sistema é avaliado por diferentes tipos de teste de desempenho:

- **Teste de carga –** com aumento contínuo da carga de trabalho.
- **Teste de estresse –** dentro ou fora dos limites da carga de trabalho prevista.
- **Teste de resistência –** com carga de trabalho contínua e significativa.
- **Teste de pico –** com uma carga de trabalho repentina e substancialmente aumentada.

Teste de Caso de Uso

O Use Case descreve como um sistema responderá a um determinado cenário criado pelo usuário que, por sua vez, se concentra nas ações e no ator, sem levar em consideração as entradas e saídas do sistema. Os desenvolvedores conduzem o teste mantendo os conceitos do projeto em foco, escrevem casos de uso e, após concluí-los, o comportamento do sistema é testado conforme o Caso de Uso especificado. Os testadores, por sua vez, os usam para criar casos de teste.

O Use Case é a técnica de teste mais usada e pode ser aplicado amplamente no desenvolvimento de testes nos níveis do sistema ou de aceitação, ajuda a descobrir os defeitos nos testes de integração, verifica se o caminho usado pelo usuário está funcionando como pretendido e garante que as tarefas possam ser realizadas com êxito. Utilizando corretamente o teste de Use Case, os responsáveis podem detectar deficiências e modificar o sistema para que ele atinja eficiência e precisão.

Teste de regressão

O teste de regressão é a prática de verificar o comportamento do *software* após as atualizações ou algum recurso implementado, para garantir que essas mudanças não tenham afetado as funções, a estabilidade e a integridade geral do sistema.

Uma **regressão de *software*** é qualquer alteração indesejada e não intencional, que ocorre a partir de alterações de código. No *software*, uma regressão pode ocorrer depois que um novo recurso é implementado. Digamos que um cálculo de preço de um determinado produto estava efetuando o cálculo corretamente. Depois de atualizações para adequar às novas legislações para o cálculo de impostos, o cálculo do preço do produto ficou errado. Dado que uma regressão é uma alteração não intencional, o teste de regressão é o processo de busca por essas alterações.

Automação de Teste

Quando os testes manuais estão mostrando testes mais precisos é sinal que a maturidade plena foi atingida e é nesse momento que esse processo necessita ser automatizado por vários motivos. Um das principais é a produtividade, pois, não alocaria uma pessoa para ficar efetuando testes repetitivos. Isso, além de desmotivar, acaba gerando um custo desnecessário. Com o ganho de tempo, facilitará o gerenciamento de todas as necessidades de teste, permitindo que mais tempo e esforço sejam gastos na criação de casos de teste eficazes.

A automatização é feita por ferramentas de automação que não requerem intervenção humana e inserem automaticamente os dados de teste, comparam os resultados esperados e reais e geram relatórios de teste detalhados.

Alguns benefícios da automatização de testes podem ser:

- Apesar de muitos benefícios, nem todos os casos são passíveis de automatização, pois, poderia não compensar o retorno de investimentos. Não se deve automatiza:
  - Casos de teste críticos para os negócios.
  - Casos de teste a serem executados repetidamente.
  - Casos de teste tediosos ou difíceis.
  - Casos de teste demorados.
  - Novos casos de teste que nem são executados manualmente nem uma vez.
  - Casos de teste com requisitos em constante mudança.

A tendência de automação de teste é suportada pela adoção cada vez maior de metodologias ágeis, que promovem práticas de automação de teste e integração contínua como o maior trunfo do desenvolvimento eficaz de software.

Alguns mitos relacionados a automatização dos testes serão encontrados em novas equipes de desenvolvimento. Dentre eles, podemos citar:

- automatização substituirá o teste manual;
- o teste manual fornece uma solução para todos os tipos de teste.

» a automação consome muito tempo;

» a automação é muito fácil;

» a automação é apenas para teste de regressão;

» o teste de automação não requer cooperação entre os testadores;

» retorna um ROI mais rápido;

» o teste de automação detecta apenas erros.

Testes de confiabilidade

"O teste de confiabilidade é um processo de teste cujo objetivo é medir a confiabilidade de um sistema. Existem diversas métricas de confiabilidade e podem ser usadas para especificar a confiabilidade requerida de software, quantitativamente. Caso o sistema alcance o nível requerido de confiabilidade, é possível verificar o processo de teste de confiabilidade" (SOMMERVILLE, 2011, p. 280).

Essa abordagem conceitualmente atrativa para a medição de confiabilidade, na prática, não é de fácil aplicação. As principais dificuldades que surgem são:

» incerteza de perfil operacional;

» custos elevados de geração de dados de teste;

» incertezas estatísticas quando alta confiabilidade é especificada;

» reconhecimento de falhas.

Teste das estruturas de controle – Teste de condição

Essa metodologia procura encontrar erros nas condições lógicas do programa:

» Erro no operador booleano (OR, AND e NOT).

» Erro de variável booleana.

» Erro de parênteses booleanos.

» Erro de operador relacional.

» Erro de expressão aritmética.

Para cada expressão relacional do tipo: E1 <operador relacional> E2, são realizados 3 testes em que a expressão E1 é maior, igual e menor do que E2.

Para expressões compostas, as expressões simples são testadas individualmente e de forma cruzada

'How to design practical test cases'. Um artigo sobre como projetar casos de teste, escrito por um autor de uma empresa japonesa que tem uma reputação muito boa em entregar softwares com pouquíssimos defeitos. (YAMAURA, T. *IEEE Software*, v. 15, n. 6, nov. 1998. Disponível em: http://dx.doi.org/10.1109/52.730835.)

How to Break Software: A Practical Guide to Testing. Esse é um livro sobre testes de software, mais prático do que teórico, no qual o autor apresenta um conjunto de diretrizes baseadas em experiências em projetar testes suscetíveis de serem eficazes na descoberta de defeitos de sistema. (WHITTAKER, J. A. *How to Break Software: A Practical Guide to Testing*. Addison-Wesley, 2002.)

'*Software Testing and Verification*'. Essa edição especial do *IBM Systems Journal* abrange uma série de artigos sobre testes, incluindo uma boa visão geral, artigos sobre métricas e automação de teste. (*IBM Systems Journal*, v. 41, n. 1, jan. 2002.)

'*Test-Driven Development*'. Essa edição especial sobre desenvolvimento dirigido a testes inclui uma boa visão geral de TDD, bem como artigos de experiência em como TDD tem sido usado para diferentes tipos de *software*. (*IEEE Software*, v. 24, n. 3, mai./jun. 2007.)

CAPÍTULO 3
Fases de Testes de validação

Como em qualquer projeto, as atividades de testes são planejadas antes do início da fase de execução. Seu principal objetivo é garantir que a equipe entenda os objetivos do cliente, do produto, os possíveis riscos que eles precisam enfrentar e os resultados que esperam alcançar. Documentos criados nessa fase, servirão para resolver essas tarefas, alinhas as atividades de teste com a finalidade geral da aplicação a ser produzida e coordenar os esforços de testes com o restante da equipe ou das equipes.

A fase de teste e validação poderá ocorrer em qualquer etapa do processo de acordo com as estratégias adotadas para o desenvolvimento do software. As estratégias podem estar relacionadas às normas ou *frameworks*, metodologias de desenvolvimento, como ágeis ou tradicionais, podem usar em conjunto com estratégias variadas e ainda, podem ser preventivas ou reativas.

Estratégias de teste

Primeiramente, é essencial que esteja muito claro para todos a definição do que será testado e quais são as métricas que deverão ser alcançadas para que cada parte, ou o todo, esteja como status de pronto. Uma boa opção para responder a essa pergunta, as equipes de controle de qualidade desenvolvem casos de teste que descreve as pré-condições, resultados desejados e pós-condições de um cenário específico, com o objetivo de verificar se são atendidos aos requisitos básicos.

Outra cultura que deve ser difundida nos ambientes de desenvolvimento – equipe de desenvolvimento, cliente etc. – é que escrever um plano é uma tarefa demorada. Nas metodologias ágeis, com foco no produto e não nos documentos, esse desperdício de tempo parece insuficiente.

Para resolver esse problema, James Whittaker, em seu artigo, podendo ser lido pelo link: https://testing.googleblog.com/2011/09/10-minute-test-plan.html, acessado em 10/1/2019, introduziu a abordagem do plano de teste de 10 minutos:

> Qualquer coisa no desenvolvimento de *software* que leve dez minutos ou menos para ser executada é trivial ou, em primeiro lugar, não vale a pena. Se vale a pena fazer, bem, isso é outra história inteiramente. Toda vez que olho para qualquer uma das dezenas de planos de teste que minhas equipes escreveram, vejo planos de teste mortos.

Segundo Whittaker, as três coisas que surgiram de mais importante, foram:

1. Atribuem os advérbios e adjetivos que descrevem os conceitos de alto nível que os testes devem garantir. Atributos como "*rápido", "utilizável", "seguro", "acessível"* e assim por diante.

2. Componentes os substantivos que definem os principais blocos de código que compõem o produto. Estas são classes, nomes de módulos e recursos do aplicativo.

3. Capacita os verbos que descrevem ações e atividades do usuário.

Como resultado, 80% concluído em 30 minutos ou menos.

Não é regra, porém, baseando-se em planejamento de projetos, podemos definir a seguinte sequência:

» análise de requisitos;

» planejamento de testes;

» desenvolvimento de casos;

» preparação de Ambiente;

» execução;

» finalização.

Existem critérios de entrada e saída associados às seis fases do ciclo de vida de testes de um software. Ao definir esses critérios, torna-se viável aos testadores concluírem a execução dos testes em um tempo fixo, mantendo a qualidade e eficiência do produto final.

Critérios de entrada

Indicam quais são os requisitos que a equipe deve se atentar para dar início aos procedimentos de teste.

Critérios de saída

Determinam os requisitos e ações que devem ser seguidos e concluídos antes do término dos testes. São destacados e identificados os defeitos de alta prioridade

que precisam ser corrigidos imediatamente, garantindo assim uma cobertura de funcionalidades completa.

Análise de Requisitos

Nessa etapa é necessário o entendimento efetivo das especificações e requisitos disponíveis, tanto requisitos funcionais, quanto requisitos não funcionais, pela equipe de testes, pois, esses requisitos produzem resultados que os alimentam com dados de entrada.

Na sequência é necessário escolher quais requisitos podem ser testáveis, pois, é muito importante que haja uma priorização, seleção e classificação dos requisitos que serão manuais ou automatizados.

Planejamento de Teste

Esse preparativo é criado para analisar os requisitos de testes necessários pela equipe do controle de qualidade. Nessa hora é desenvolvido um escopo e definidos os objetivos baseando-se no total entendimento do contexto e domínio do produto de software. A criação do planejamento é caracterizada na gestão de qualquer projeto, onde são criados o gerenciamento de riscos, gerenciamento de custos, gerenciamentos de prazos etc. Umas das únicas diferenças é a definição do ambiente de testes. Como resultado, deverá ser criada documentação do plano de testes e uma estimativa de tempo, custo e recursos.

Desenvolvimento de Caso de Teste

O principal objetivo das técnicas de design de caso de teste é testar as funcionalidades e recursos do *software* com a ajuda de casos de teste eficazes. As técnicas de design de caso de teste são amplamente classificadas em três categorias principais: caixa preta, caixa branca e baseado na experiência da equipe.

Na sequência, vem a verificação e validação dos requisitos que foram ou deveriam ser especificados no estágio de documentação, onde deverá conter, também, a revisão, atualização e aprovação de scripts de automação e casos de teste são processos essenciais desse estágio.

Preparação do ambiente

A preparação do ambiente de teste cabe ao gerente de controle de qualidade supervisionar que irá cuidar de sua montagem e configuração. Ele precisa garantir que o ambiente de teste esteja o mais próximo possível do ambiente real do usuário final, como servidores, estruturas, hardware e *software* - para a execução de casos de teste desenvolvidos. A configuração de *software* e *hardware*, juntamente com a configuração dos dados de teste, são os principais componentes desta fase. Ferramentas como contêineres são utilizadas para garantir o mesmo ambiente para todos os mesmos ambientes.

Execução de Teste

Nessa etapa os testadores executam os casos de testes sob um produto de *software* provisoriamente definido como pronto. Os testadores irão identificar, relatar e registrar todos os defeitos e erros, além de comparar os resultados esperado com o teste real.

Quando a equipe de desenvolvimento remover um erro, o teste de regressão é iniciado. O teste de regressão é para garantir que o *software* ou aplicativo funcione mesmo após a implantação de uma alteração.

Encerramento do teste

Essa é a última etapa do ciclo de vida do planejamento de testes. A execução do teste e a entrega do produto final marcam o início da fase de encerramento. A equipe de controle de qualidade verifica os resultados do teste e os discute com outros membros da equipe. Alguns outros fatores que eles consideram importantes discutir é a qualidade do produto, a cobertura do teste e o custo do projeto, as métricas de teste, o cumprimento das metas e a aderência aos prazos. Uma vez que tenham uma compreensão total do que aconteceu, eles podem avaliar toda a estratégia e processo de teste.

Segundo Sommerville (2011, p. 275), "os processos de validação devem demonstrar que o sistema atende a sua especificação e que os serviços e o comportamento do sistema suportam os requisitos do cliente, pois, geralmente, descobrem erros de requisitos e projeto e defeitos de programas que precisam ser reparados".

São duas as razões para tanto:

Custos de falhas

Os custos de falhas de sistemas críticos são mais altos do que os de sistemas não críticos. Somente com a verificação completa do sistema e sua validação é possível reduzir tais custos. É muito mais econômico encontrar e corrigir defeitos antes que uma vez entregue o sistema, ele ocasione acidentes ou interrupções de sistemas que signifiquem prejuízo e custos extras para o cliente final.

Validação de atributos de confiança

Demonstra que o sistema atende a requisitos de confiança como disponibilidade, confiabilidade e proteção através de exemplos formais que certificam a segurança do sistema antes de sua implantação.

Verificação Estatística

A análise estatística possui técnicas de verificação de sistemas que não envolvem a execução de um programa. Elas possibilitam uma representação do *software* e de seus modelos de projeto ou código-fonte do programa. Através dessas técnicas é possível encontrar erros antes que uma versão executável do sistema já esteja disponível e com a vantagem de que a presença dos erros não impede a verificação do sistema.

Verificação de Modelos

Utiliza ferramentas de modelagem de projetos para verificar anomalias na UML, ou seja, um mesmo nome sendo usado para objetos diferentes. Para sistemas críticos, outras técnicas de análise estática também podem ser utilizadas, tais como:

1. Verificação Formal: produz matematicamente argumentos para que um programa siga e atenda às suas especificações.
2. Verificação de Modelos: utiliza um provador de teoremas para verificar se existem inconsistências no sistema.
3. Análise Automatizada de Programa: utiliza padrões conhecidos de erros potenciais para examinar o código-fonte de um software.

CAPÍTULO 4
Aceite

Considerada a última etapa do processo, o teste de aceitação é o detalhamento formal do comportamento de um produto de software, geralmente expresso como exemplo ou cenário de uso que, durante todo o processo, várias notações e abordagens diferentes foram propostas para esses exemplos ou cenários.

Segundo Sommerville (2011, p. 28), “esse é o estágio final do processo de testes, antes que o sistema seja aceito para uso operacional. ”O sistema passa por testes realizados a partir de dados fornecidos pelo cliente e esses testes de aceitação podem revelar erros e omissões no levantamento de requisitos do software. Os dados reais colocam o sistema em funcionamento de formas diferentes dos dados de testes, o que contribui para averiguar se de fato os recursos do sistema atendem às necessidades dos usuários ou se o desempenho do sistema é inaceitável.

Embora pequenos problemas deveriam ter sido detectados e resolvidos no início do processo, nessa etapa de teste se concentra na qualidade geral do sistema, desde conteúdo e interface do usuário até problemas de desempenho ou até mesmo, defeitos. Os defeitos podem impactar amplamente a continuidade dos negócios, mas o gerente de projetos não deve esquecer que os defeitos também podem impactar os objetivos do projeto.

Se defeitos e deficiências precisarem ser corrigidos, significa que a equipe do projeto gastará mais esforço e tempo no projeto. É importante avaliar essas possibilidades desde o início do projeto e definir o plano certo para apoiar questões como essa com o desvio mínimo das metas iniciais.

O estágio de aceitação pode ser seguido por um teste alfa e beta que permita a experimentação do *software* por um número pequeno de usuários reais antes que ele seja implantado e lançado.

Um grande equívoco é quando os critérios de aceitação não são definidos no início do projeto – ou mesmo se forem, eles são descartados. Geralmente, as equipes acreditam que tem o mesmo ponto de vista do usuário, em relação ao domínio do produto, porém, sem nada documentado. Portanto, é muito importante que a documentação esteja, além de alinhada, ilustrada e documentada.

Durante o processo de desenvolvimento as especificações podem sofre alterações. Mesmo se isso acontecer, os critérios de aceitação devem ser descritos no início do

projeto para configurar os critérios de saída desde o início. E, pouco antes de iniciar o processo de aceitação do usuário, a equipe deve concordar sobre quais casos de teste devem ser executados e quais são os resultados esperados. Isso ajudará a alinhar seus critérios com o usuário final.

O ponto principal é ter o mesmo conjunto de casos de teste que o usuário final, quando for ser executado o teste de aceitação. Se o usuário final estiver gerenciando sua própria lista de casos de teste, surgirão muitos problemas desagradáveis, pois a visão dos desenvolvedores e testadores poderão ser outra, causando sérios problemas como atraso em cronograma, estouro de orçamento etc.

O teste de aceitação é uma técnica de teste de caixa preta em que apenas as funcionalidades são verificadas para garantir que o produto atenda aos critérios de aceitação especificados.

Mesmo que o teste do sistema tenha sido concluído com êxito, o teste de aceitação é exigido pelo cliente ou, se o cliente não o exigir, isso deverá ser uma etapa obrigatória no processo de testes.

A importância se dá para:

- Ganhar confiança no produto que está sendo lançado no mercado.
- Para garantir que o produto esteja funcionando da maneira que precisa.
- Garantir que o produto corresponda aos padrões atuais do mercado e seja competitivo o suficiente com os outros produtos similares no mercado.

Tipos de Testes de Aceitação

Teste de aceitação do usuário (UAT – User Acceptance Test)

Também conhecido como teste de usuário final, esse teste verifica-se o Produto está funcionando corretamente para o usuário, seguindo os requisitos específicos, frequentemente usados pelos usuários finais, principalmente para fins de teste. O termo “Usuário” aqui significa os usuários finais a quem a aplicação é destinada e, portanto, o teste é realizado da perspectiva dos usuários finais e do ponto de vista deles.

Teste de aceitação de negócios (BAT – Business Acceptance Test)

Esse tipo de teste avalia se o Produto atende ou não aos objetivos de negócios, principalmente nos benefícios comerciais que são bastante desafiadores devido às mudanças nas condições do mercado/tecnologias avançadas, para que a implementação atual possa ter que sofrer alterações que resultem em orçamentos extras. Isso pode levar o produto a falhar, mesmo passando pelos requisitos técnicos.

Teste de aceitação de contrato (CAT - Contract Acceptance Test)

O contrato assinado aqui é denominado Acordo de Nível de Serviço (SLA), que inclui os termos de período de teste, áreas de teste, condições sobre problemas encontrados em estágios posteriores, pagamentos etc., serão realizados, o que significa que o contrato será cumprido.

Testes de aceitação de conformidade (RAT - Regulation Acceptance Test)

O Teste de aceitação de regulamentos, também conhecido como Teste de aceitação de conformidade, examina se o *software* está em conformidade com os regulamentos. Isso inclui regulamentos governamentais e legais. Se alguma das regras e regulamentos for violada em qualquer país, esse país ou a região específica desse país não poderá usar o Produto e será considerado uma Falha. Os fornecedores do Produto serão diretamente responsáveis se o Produto for lançado, mesmo que exista uma violação.

Teste de aceitação operacional (OAT - Operation Acceptance Test)

Esse tipo de teste garante a existência de fluxos de trabalho para permitir que o *software* ou sistema seja usado. Isso deve incluir fluxos de trabalho para planos de backup, treinamento do usuário e vários processos de manutenção e verificações de segurança.

Isso é para avaliar a prontidão operacional do Produto e é um teste não funcional. Inclui principalmente testes de recuperação, compatibilidade, manutenção, disponibilidade de suporte técnico, confiabilidade, *failover*, localização etc.

Teste alfa

O teste alfa normalmente ocorre no ambiente de desenvolvimento e geralmente é feito pela equipe interna. Muito antes de o produto ser lançado para testadores ou clientes externos. Grupos de usuários em potencial também podem realizar testes Alpha, mas o importante aqui é que eles ocorrem no ambiente de desenvolvimento. Com base no feedback – coletado dos testadores alfa –, as equipes de desenvolvimento corrigem certos problemas e melhoram a usabilidade do produto.

Teste beta

O teste beta ou "teste de campo", ocorre no ambiente do cliente e envolve alguns testes extensivos por um grupo de clientes que usam o sistema em seu ambiente. Esses testadores beta fornecem feedback, o que, por sua vez, leva a melhorias no produto.

Extreme Programming

Tanto o teste de aceitação como o desenvolvimento em Extreme Programming são incrementais. O cliente é parte integrante da equipe e escreve os testes enquanto o desenvolvimento continua. Assim, todos os novos códigos são validados, garantindo que resultam realmente nas funcionalidades das quais o cliente necessita. (SOMMERVILLE, p. 48).

Nos métodos ágeis, como XP, o teste de aceitação prevê que são os usuários que devem decidir quando o sistema é aceitável. No XP, o usuário é parte da equipe de desenvolvimento como um testador alfa, fornecendo os requisitos de sistema. Os usuários também são responsáveis pela definição dos testes e se eles são condizentes com a história dos usuários. (SOMMERVILLE, p. 161).

Testes de usuário

São um estágio do processo de testes no qual os usuários fornecem conselhos e instruções sobre os testes do sistema. Pode envolver testes formais no caso de um sistema que seja aprovado por fornecedor externo ou processos informais nos quais os usuários experimentam o *software* para averiguar se o produto faz o que eles realmente necessitam. O teste de usuário é essencial tanto para sistemas abrangentes como para os mais simples, pois o ambiente de trabalho dos usuários oferece influências significativas e efeitos importantes sobre a confiabilidade, desempenho, segurança e usabilidade de um sistema.

Teste de Aceitação

O teste de aceitação é essencial no desenvolvimento de sistemas customizados, pois é através dele que o cliente decide se o sistema desenvolvido deve ser aceito ou não. A aceitação significa que o pagamento pelo sistema pode ser realizado. São seis os estágios do processo de testes de aceitação:

1. Definição dos critérios de aceitação.
2. Planejamento dos testes de aceitação.
3. Derivação dos testes de aceitação.
4. Execução dos testes de aceitação.
5. Negociação dos resultados dos testes.
6. Aceitação ou Rejeição do Sistema.

GERENCIAMENTO DE TESTES

UNIDADE III

CAPÍTULO 1
Conceitos

Nesse capítulo, serão apresentados os conceitos básicos para a realização do gerenciamento de teste. Por se tratar um planejamento ou projeto, muitas etapas de gerenciamento estarão ligadas ao conceito de boas práticas, como o PMBOK. Por exemplo, uma sugestão de processos de gerenciamento de testes, podem seguir as seguintes etapas:

- **Planejamento**
  - Gerenciamento de risco.
  - Estimativa de teste.
  - Planejamento de teste.
  - Organização de Teste.
- **Execução**
  - Monitoramento e controle de teste.
  - Gerenciamento de problemas.
  - Relatório de teste e avaliação.

Planejamento

Gerenciamento de Risco

Segundo o PMBOK, planejar o gerenciamento dos riscos é o processo de definição de como conduzir as atividades de gerenciamento dos riscos de um. O planejamento cuidadoso e explícito aumenta a probabilidade de sucesso para os outros cinco processos de gerenciamento dos riscos. O planejamento dos processos de gerenciamento dos riscos é importante para garantir que o grau, o tipo e a visibilidade do gerenciamento dos riscos sejam proporcionais tanto aos riscos como à importância do projeto para a organização. O planejamento também é importante para fornecer tempo e recursos suficientes para as atividades de gerenciamento dos riscos e para estabelecer uma base acordada para a avaliação dos riscos.

Risco é a perda potencial (um resultado indesejável, mas não necessariamente) resultante de uma determinada ação ou atividade.

Estimativa de teste

A estimativa do esforço para o teste é uma das tarefas principais e importantes no Gerenciamento de Testes. Uma estimativa é uma previsão que dirá se o teste está aproximadamente determinando quanto tempo uma tarefa levaria para ser concluída. Estimativas precisas de teste levam a um melhor planejamento, execução e monitoramento de tarefas sob a atenção de um gerente de teste.

Planejamento de teste

Um plano de teste pode ser definido como um documento que descreve o escopo, a abordagem, os recursos e o cronograma das atividades de teste pretendidas. O plano está bem definido em normas como a IEEE 820 e será detalhado no capítulo 4 da Unidade IV. Um projeto pode falhar sem um plano de teste completo.

Nos testes de software, um plano de teste fornece informações detalhadas sobre um esforço de teste futuro, incluindo:

- Estratégia de Teste.
- Objetivo do teste.
- Critérios de saída/suspensão.

» Planejamento de recursos.

» Entregas de teste.

O planejamento de teste é essencial no desenvolvimento de grandes sistemas de software.

Organização de Teste

Logo após a elaboração do plano de teste, a melhor forma de executá-lo é se ater a fase de organização do teste. Genericamente, é preciso organizar uma equipe de teste eficaz, qualificada para executar o mecanismo de teste sempre crescente de maneira organizada.

Execução

Monitoramento e controle de teste

O Monitoramento e Controle de Teste é o processo de supervisionar todas as métricas necessárias para garantir que o projeto esteja funcionando bem, dentro do cronograma e dentro do orçamento.

Monitorar é um processo de coletar, registrar e relatar informações sobre a atividade do projeto que o gerente do projeto e as partes interessadas precisam conhecer.

Para monitorar, o Gerente de Testes realiza as seguintes atividades:

» definir a meta do projeto, padrão de desempenho do projeto, baseando em métricas;

» observa e compara as expectativas de desempenho reais e as planejadas;

» registra e relata qualquer anomalia detectada que aconteça com o projeto.

O Controle de Projeto é um processo que utilizam os dados, compilados na atividade de monitoramento para levar o desempenho real ao desempenho planejado. Nesta etapa, o Gerenciador de Testes executa ações para corrigir os desvios do plano. Em alguns casos, o plano deve ser ajustado de acordo com a situação do projeto.

Gerenciamento de problemas

Conforme mencionado no início dos tópicos, todos os projetos podem ter riscos potenciais, ou seja, pode acontecer algo de anormal com o produto de software. Quando o risco acontece, torna-se um problema. No ciclo de vida de qualquer projeto, sempre haverá problemas e perguntas inesperadas que podem colocar em dúvida a capacidade das habilidades da equipe, ou colocar em dúvidas o cumprimento dos prazos e dos custos.

Relatório de teste e avaliação

Esse relatório descreve os resultados do Teste em termos de cobertura de teste e critérios de saída. Os dados usados na Avaliação de Teste são baseados nos dados dos resultados do teste e no resumo do resultado do teste.

Responsáveis pelo gerenciamento de testes

Por se tratar de um time, muitas vezes organizados em metodologias que favoreçam o entrosamento, como o DevOps, a responsabilidade geral pelo bom desempenho do gerenciamento de testes, é, sem dúvida, de toda equipe. Entretanto, mesmo nas equipes ágeis, existirá o papel do facilitador que, chama para si, a reponsabilidade por estar mais próximo ao cliente. Ele atua como suporte de teste e está envolvido no planejamento e gerenciamento de recursos e na resolução de problemas que criam um obstáculo no esforço de teste.

O líder ou gerente de teste é responsável por:

- liderando a equipe de teste.
- delineando o escopo do teste.
- dedicar e gerenciar recursos de teste.
- aplicando métricas de teste.
- agendamento, instalação e manipulação do esforço de teste.
- compreensão completa de seu papel na organização.

Documentação de Testes

De acordo com o padrão IEEE para documentação de teste de software, um documento do plano de teste deve conter as seguintes informações:

- Identificador do plano de teste.
- Introdução.
- Referências (lista de documentos relacionados).
- Itens de teste (o produto e suas versões).
- Recursos a serem testados.
- Recursos a não serem testados.
- Critérios de aprovação ou reprovação de itens.
- Abordagem de teste (níveis, tipos, técnicas de teste).
- Critérios de suspensão.
- Entregas (plano de teste (este documento em si), casos de teste, scripts de teste, logs de defeitos/aprimoramentos, relatórios de teste).
- Ambiente de teste (*hardware*, *software*, ferramentas).
- Estimativas.
- Cronograma.
- Necessidades de pessoal e treinamento.
- Responsabilidades.
- Riscos.
- Pressupostos e Dependências.
- Aprovações.

Os padrões para documentação de teste se enquadram no IEEE 829-2008, também conhecido como 829 Standards para documentação de teste de *software* e sistema. O padrão define uma maneira de realizar atividades de teste, ou seja, abrange cada fase

do ciclo de vida de teste de software. Os padrões são definidos de forma a atender às diversas necessidades de diferentes organizações que desejam implementar o padrão.

O IEEE declara oito estágios do processo de documentação, com cada estágio tendo seu próprio documento separado.

» **Gerenciamento de Testes**

Contém a descrição detalhada sobre como o processo de teste deve prosseguir. Isso inclui informações como quanto tempo será alocado para o teste, quem o realizará, o que será testado e a qualidade a ser alcançada no final do processo de teste.

» **Especificação do projeto de teste**

Inclui as condições de teste a serem implementadas e seus resultados.

» **Especificação de Caso de Teste**

Informações sobre dados específicos que precisam ser testados, com base nos dados coletados anteriormente.

» **Especificação do procedimento de teste**

Específica como exatamente o testador deve executar um teste, o ambiente/ configuração necessário para executar os testes.

» **Relatório de transmissão de itens de teste**

Este é o relatório sobre a transferência de itens testados de um estágio para o seguinte.

» **Log de teste**

Contém os detalhes dos testes executados, a sequência na qual eles foram executados e o relatório de aprovação/reprovação dos testes.

» **Relatório de Incidentes de Teste**

Pontos principais que indicam o motivo do desvio nos resultados reais e esperados.

» **Relatório de resumo do teste**

Um relatório completo do resumo, indicando o procedimento geral do teste e o resultado, como a eficiência com que o teste foi realizado, avaliando a qualidade do sistema, o tempo necessário para testar uma determinada condição e se houve algum problema ou não.

A norma específica a preparação dos seguintes tipos de documentos, para que possamos obter uma quantidade adequada de informações sobre o processo de teste.

Plano de teste principal

Descreve o plano geral de teste em detalhes em vários níveis de teste. Tendo em vista os requisitos de *software* e o planejamento de garantia de qualidade (abrangente) do projeto, o planejamento mestre de teste como uma atividade compreende selecionar as partes constituintes do esforço de teste do projeto; estabelecendo os objetivos para cada parte; estabelecer a divisão do trabalho (tempo, recursos) e as inter-relações entre os partes; identificar os riscos, premissas e padrões de fabricação a serem considerados e contabilizado pelas partes; definir os controles do esforço de teste; e confirmando os objetivos aplicáveis definido pelo planejamento da garantia de qualidade. Identifica o esquema do nível de integridade e o nível de integridade selecionado, o número de níveis de teste, as tarefas gerais a serem executadas e a documentação requisitos.

Acompanha a descrição completa de um MTP:

1. Introdução

Esta seção identifica o documento e o coloca em contexto do ciclo de vida específico do projeto. É nesta seção que todo o esforço de teste é descrito, incluindo a organização de teste, o cronograma de teste e o esquema de integridade. Um resumo dos requisitos recursos, responsabilidades, ferramentas e técnicas também podem ser incluídos nesta seção.

1.1 Identificador de documento

Identifique exclusivamente uma versão do documento, incluindo informações como a data de emissão, a organização emissora, autor(es), assinaturas de aprovação (possivelmente eletrônicas) e status/versão (por exemplo, rascunho, revisado, corrigido ou final). As informações de identificação também podem incluir os revisores e gerentes pertinentes.

1.2 Escopo

Descreve o objetivo, as metas e o escopo do esforço de teste do sistema/software.

1.3 Referências

Listar todos os documentos de referência aplicáveis. As referências são separadas em referências “externas” impostos externamente ao projeto:

a) Leis.

b) Regulamentos governamentais.

c) Padrões (por exemplo, governo e / ou consenso).

d) Políticas.

Referências “internas” impostas de dentro para o projeto:

a) Autorização do projeto.

b) Plano do projeto (ou plano de gerenciamento do projeto).

c) Plano de garantia da qualidade.

d) Plano de gerenciamento da configuração.

Isso também pode estar no final do documento.

1.4 Visão geral do sistema e principais recursos

Descreve a missão ou o objetivo comercial do sistema ou produto de *software* em teste (ou referência onde as informações podem ser encontradas, por exemplo, em um documento de definição do sistema, como um Conceito de Operações).

1.5 Visão geral do teste

Descreva a organização de teste, o cronograma de testes, o esquema de nível de integridade, os recursos de teste, as responsabilidades, ferramentas, técnicas e métodos necessários para executar o teste.

1.5.1 Organização

Descreva a relação dos processos de teste com outros processos, como desenvolvimento, projeto gerenciamento, garantia de qualidade e gerenciamento de configuração. Inclua as linhas de comunicação dentro da (s) organização (s) de teste, a autoridade para resolver

problemas levantados pelas tarefas de teste e as autoridades para aprovar produtos e processos de teste. Isso pode incluir (mas não deve se limitar a) uma representação visual, por exemplo, um organograma.

1.5.2 Programação de teste mestre

Descreva as atividades de teste no ciclo de vida e nos marcos do projeto. Resuma o cronograma geral das tarefas de teste, identificando onde os resultados da tarefa retornam ao desenvolvimento, organização e processos de suporte (por exemplo, garantia de qualidade e gerenciamento de configuração). Descreva a tarefa política de iteração para a reexecução de tarefas de teste e quaisquer dependências.

1.5.3 Esquema de nível de integridade

Descreva o esquema de nível de integridade identificado para o sistema ou produto de *software* baseado em *software* e o mapeamento do esquema selecionado para o esquema de nível de integridade usados nesta norma.

1.5.4 Resumo dos recursos

Resuma os recursos de teste, incluindo equipe, instalações, ferramentas e requisitos processuais especiais.

1.5.5 Responsabilidades

Forneça uma visão geral dos tópicos e responsabilidades do conteúdo organizacional para testar tarefas. Identifique os componentes organizacionais e suas principais e secundárias responsabilidades relacionadas ao teste.

1.5.6 Ferramentas, técnicas, métodos e métricas

Descreva documentos, hardware e software, ferramentas de teste, técnicas, métodos e ambiente de teste para ser usado no processo de teste. Descreva as técnicas que serão usadas para identificar e capturar reutilizáveis *testware*. Inclua informações sobre aquisição, treinamento, suporte e qualificação para cada ferramenta, tecnologia e método.

2. Detalhes do plano de teste principal

Introduza as seguintes seções subordinadas. Esta seção descreve os processos de teste, teste requisitos de documentação e requisitos de relatório de teste para todo o esforço de teste.

2.1 Processos de teste, incluindo definição de níveis de teste

Identifique as atividades e tarefas de teste a serem executadas para cada um dos processos de teste descritos nesta norma ou os processos de teste alternativos definidos pelo usuário deste padrão) e documentar os atividades e tarefas de teste.

2.1.1 Processos: Gerenciamento

Descreva como todos os requisitos da norma são atendidos (por exemplo, fazendo referência cruzada a esta norma) se o ciclo de vida usado no MTP for diferente do modelo de ciclo de vida neste padrão. O teste requer planejamento antecipado que abrange várias atividades de desenvolvimento.

Aborde os tópicos a seguir para cada atividade de teste:

2.1.1.1 Atividades: Gerenciamento do esforço de teste

A atividade de gerenciamento de teste monitora e avalia todos os resultados do teste. A atividade de gerenciamento de teste é realizada em todos os processos e atividades do ciclo de vida. Esta atividade analisa continuamente os testes, gera o MTP, se exigido pelo nível de integridade, e revisa o MTP conforme necessário.

2.1.2 Processos: aquisição

O processo de aquisição começa com a definição da necessidade de adquirir um sistema, produto de *software* ou serviço de software. Termina com o gerenciamento do processo de aquisição até a aceitação do sistema, produto de *software* ou serviço de software. As tarefas de teste são executadas durante todo o processo de aquisição, a fim de apoie isso.

2.1.2.1: Atividade: Teste de suporte à aquisição

No início do processo de aquisição, descrito acima, a atividade de teste executa várias tarefas que apoiam o início do projeto, especialmente a solicitação de proposta e a preparação de contratos.

2.1.3 Processos: fornecimento

O processo de fornecimento é iniciado por uma decisão de preparar uma proposta para responder à RFP de um adquirente ou por assinatura e celebração de um contrato com o adquirente para fornecer o sistema baseado em software, produto ou serviço.

2.1.3.1 Atividades: teste de planejamento

Os participantes da atividade de planejamento de teste podem participar do início de uma solicitação de proposta, preparação da resposta, planejamento de contratos, execução e controle, revisão e avaliação e entrega e atividades de conclusão.

2.1.4 Processos: Desenvolvimento

O processo de desenvolvimento consiste nas atividades e tarefas do grupo de desenvolvimento. O processo abrange as atividades de desenvolvimento de análise de requisitos, design, codificação, integração de componentes, teste, instalação e aceitação, relacionados ao *software* ou ao produto do sistema baseado em software.

2.1.4.1 Atividades: conceito

A atividade conceitual representa a identificação de uma implementação específica para resolver as partes interessadas problema ou preencha as necessidades das partes interessadas.

2.1.4.2 Atividades: Requisitos

A atividade de requisitos define os funcionais e os não funcionais, seguidos por interfaces externas ao software, segurança e requisitos de segurança, definições de dados, documentação do usuário para o sistema e software, instalação e requisitos de aceitação, requisitos de operações e execução do usuário e requisitos de manutenção do usuário.

2.1.4.3 Atividades: Design

Na atividade de design, os requisitos do sistema são transformados em arquitetura e design detalhado para cada componente de software.

2.1.4.4 Atividades: Implementação

A atividade de implementação implementa o design em código, estruturas de banco de dados e representações executáveis em máquinas relacionadas.

2.1.4.5 Atividades: teste

A atividade de teste abrange a integração de componentes de software, teste de aceitação de software, integração de sistemas, e teste de aceitação do sistema. O objetivo do esforço de teste é verificar se os requisitos de *software* e os requisitos do sistema são atendidos pela execução de testes de integração, sistema e aceitação de componentes.

2.1.4.6 Atividades: instalação / checkout

A atividade de instalação/checkout abrange a instalação do sistema baseado em software, *software* produto ou serviço no ambiente de destino e a revisão e teste de aceitação do adquirente do produto.

2.1.5 Processos: Operação

O processo de operação abrange a operação do produto de *software* e o suporte operacional aos usuários. A atividade de operação realiza testes operacionais, operação do sistema e suporte ao usuário.

2.1.5.1 Atividades: teste operacional

A atividade de teste operacional é o uso do sistema, produto ou serviço baseado em *software* até o final usuário em um ambiente operacional. A atividade de teste operacional realiza teste operacional, sistema operação e suporte ao usuário.

2.1.6 Processos: manutenção

O processo de manutenção é ativado quando o sistema ou produto de *software* baseado em *software* sofre modificações no código e na documentação associada causadas por um problema, necessidade de aprimoramento ou adaptação.

2.1.6.1 Atividades: teste de manutenção

A atividade de manutenção abrange modificações (código, configuração, ambiente etc.), migração e aposentadoria do sistema ou produto de *software* baseado em software.

2.2 Requisitos de documentação de teste

Defina a finalidade, o formato e o conteúdo de todos os outros documentos de teste que serão usados.

2.3 Requisitos de administração de teste

Descreva os processos de resolução e geração de relatórios de anomalias, política de iteração de tarefas, política de desvio, procedimentos e padrões de controle, práticas e convenções. Essas atividades são necessárias para administrar os testes durante a execução.

2.4 Requisitos de relatório de teste

Especifique a finalidade, o conteúdo, o formato, os destinatários e o tempo de todos os relatórios de teste. O relatório de teste consiste em Logs de teste, Relatórios de anomalias Relatórios intermediários de status de teste intermediário, Relatório(s) de Teste de Nível e Relatório de Teste Mestre. Os relatórios de teste também podem incluir relatórios opcionais definidos pelo usuário deste padrão. O formato e agrupamento do opcional os relatórios são definidos pelo usuário e variam de acordo com o assunto.

3. Geral

Introduza as seguintes seções subordinadas. Esta seção inclui o glossário de termos e siglas. Também descreve a frequência e o processo pelo qual o MTP é alterado e baselined. Também pode conter uma página de alteração que contém o histórico das alterações (data, motivo da mudança e quem iniciou a mudança).

3.1 Glossários

Forneça uma lista alfabética de termos que podem exigir definição para os usuários do MTP com suas definições correspondentes. Isso inclui siglas. Também pode haver uma referência a um glossário de projeto, possivelmente publicado online.

3.2 Documentar procedimentos e histórico de alterações

Especifique os meios para identificar, aprovar, implementar e registrar alterações no MTP. Este pode ser gravado em um sistema geral de gerenciamento de configuração documentado em um Plano de Gerenciamento mencionado aqui. Os procedimentos de mudança precisam incluir um log de as alterações que ocorreram desde o início do MTP.

CAPÍTULO 2
Gestão de ferramentas

Quando se fala em ferramentas, a automatização de testes é a primeira que vem à cabeça. Isso acontece porque antes desse processo, todo trabalho é feito manualmente. Entretanto, o sucesso em qualquer automação de teste depende da identificação da ferramenta certa, ou seja, selecionar a Ferramenta de Teste "correta" para o projeto é uma das melhores maneiras de atingir o objetivo do projeto.

A Análise de Viabilidade de Automação é um fator muito significativo nos testes. Nesta análise, é preciso verificar se o aplicativo em teste está qualificado para teste automatizado. Nesta análise, é preciso verificar se o aplicativo em teste está qualificado para teste automatizado.

Para tanto, existem muitos tipos de ferramentas de teste, que o Gestor ou Gerente de Testes pode considerar ao selecionar.

As ferramentas mais utilizadas são as ferramentas de código fonte aberto. Ferramentas de código aberto são o programa em que o código-fonte é publicado abertamente para uso e/ou modificação, gratuitamente. Essas ferramentas estão disponíveis desde o gerenciamento de casos de teste até o rastreamento de defeitos.

Já as ferramentas comerciais são os *softwares* produzidos para venda ou para fins comerciais. Elas têm mais suporte e mais recursos de um fornecedor do que as ferramentas de código aberto.

Em alguns projetos de teste, o ambiente de teste e o processo de teste têm características especiais. Quando nenhuma ferramenta comercial ou de código aberto pode atender ao requisito é preciso considerar o desenvolvimento da ferramenta customizada.

Para ter a certeza que a ferramenta será viável àquele projeto, os requisitos da ferramenta devem ser avaliados com muita precisão. Para isso, todo requisito deve ser documentado e revisado pelas equipes do projeto e pelo conselho de administração.

Para avaliar as ferramentas é preciso criar uma lista de ferramentas que melhor atenda aos critérios de projeto. Um fator que se deve considerar são os fornecedores, a reputação do fornecedor, o suporte pós-venda, a frequência de atualização da ferramenta etc. ao tomar sua decisão. Muitos fornecedores disponibilizam testes gratuitos nas ferramentas.

Algumas ferramentas que podem auxiliar no teste de software. A lista poderá ser consultada em: http://www.aprendendotestar.com.br/ferramentas.

QA Complete

QA complete oferece experiência de nível empresarial, mas é flexível o suficiente para se adequar a qualquer metodologia de desenvolvimento moderna como Agile e DevOps. Esta ferramenta de gerenciamento de testes permite a ligação de testes manuais e automatizados para defeitos, requisito e tarefas.

Funcionalidades:

- É possível priorizar esforço de testes e identificar problemas de alto risco.
- Determina a cobertura do teste e assegura que existam testes para todos os requisitos
- Programe testes automatizados
- Integra com Jenkins, JIRA, Selenium e mais de 40 ferramentas.
- Configura e monitora acordos de nível de serviço (SLA)

Link: https://smartbear.com/product/qacomplete/overview/.

Qtest

Qtest da QASymphony é uma plataforma de testes construída para equipes corporativas praticando Agile e DevOps. Esta plataforma tem UI moderna, baseada em navegação que facilita todas as atividades de teste de gerenciamento de testes, automação e relatórios. Qtest também tem integração com ferramentas de desenvolvimento como Jira Software, Jenkins, e GitHub para rastreabilidade.

Funcionalidades:

- Integração em tempo real com Jira.
- Relatórios sólidos e análises.
- Gerenciamento de testes Agile.
- Exploratórios e testes baseados em sessão.
- BDD.

Link: https://www.qasymphony.com/software-testing-tools/.

TestLink

TestLink é uma ferramenta de gerenciamento de teste web que facilita a garantia de qualidade de software. A ferramenta oferece suporte para casos de teste, planos de teste, gerenciamento de usuários, bem como vários tipos de relatórios e estatísticas.

Funcionalidades:

- Fácil exportação e importação de casos de teste.
- Integra com muitas ferramentas de gerenciamento de defeito como mantis, bugzilla e redmine.
- Método fácil para a atribuição de casos de teste para vários usuários.

Link: http://testlink.org/.

Zephyr

Zephyr para JIRA oferece um total de soluções de gerenciamento de teste em destaque. Nesta ferramenta, o teste é integrado no ciclo de projeto que permite ao testador monitorar a qualidade do *software* e tomar decisões. A ferramenta também permite que o fluxo de processos possa ser flexível para fazer mudanças mais rápidas.

Funcionalidades:

- criar, ver, modificar e executar testes;
- ajuda a planejar estratégia de execução de teste;
- integração Agile com casos de teste;
- recurso de marcador visual para manter o controle de posição a execução do teste;
- ZQL Idioma para avançado busca.

Link: https://www.getzephyr.com/.

Loadrunner

É uma ferramenta de teste de carga para Windows e Linux, que permite testar a aplicação web de forma eficiente. É ferramenta de testes útil para determinar o desempenho e resultado da aplicação web sob carga pesada.

Funcionalidades:

» Oferece suporte para vários tipos de Aplicativos.

» Esta ferramenta de teste pode trabalhar em vários ambientes corporativos.

» Todos os usuários podem ser controlados com apenas um único painel.

» Fornece suporte para vários tipos de protocolos.

» O monitoramento e análise é muito usual e fácil de entender.

Link: https://saas.hpe.com/en-us/software/loadrunner/try-now.

LoadUI Pro

LoadUI é uma ferramenta de teste de carga fonte aberta que permite a realização de testes de carga complexos arrastando os diferentes componentes. Ele também permite criar e atualizar os casos de teste enquanto ele está executando, que o tornam uma ferramenta digna de usar.

Funcionalidades:

» Permite criar várias estratégias de desempenho.

» Reutilização de testes funcionais Pro SoapUI existentes.

» Feedback em tempo real sobre os resultados dos testes de carga.

» Teste de carga simultâneo mesmo nos cenários complexos.

Link: https://www.loadui.org/downloads/download-loadui-pro.html.

Jmeter

JMeter é uma ferramenta de teste de carga aberto. É uma aplicação desktop Java, projetado para carregar o comportamento funcional de teste e medir o desempenho de websites. A ferramenta foi desenvolvida com a finalidade de aplicações web de teste de carga, mas agora é expandido para outras funções de teste

Funcionalidades:

» JMeter permite realizar carga e teste de desempenho para vários tipos de servidor.

» Permite aos usuários para gerar o plano de teste usando um editor de texto.

» A ferramenta pode ser utilizada para realizar testes automatizados e funcional das aplicações.

Link: http://jmeter.apache.org/download_jmeter.cgi.

BlazeMeter

BlazeMeter é uma ferramenta de teste de carga que assegura a entrega de *software* de alto desempenho para executar rapidamente testes de desempenho para aplicativos móveis, site ou API para verificar o desempenho em todas as fases de seu desenvolvimento.

Funcionalidades:

» Permite testar o site e integrar várias localidades com resultados em relatório único.

» Recuperar dados do site da conta do Google Analytics e integrá-los em uma nova configuração de teste.

» Use credenciais VPN para integrar uma série de servidores de carga na rede privada.

Link: http://info.blazemeter.com/live-request-a-demo.

Selenium

O Selenium é a mais popular ferramenta de testes automatizados. É especificamente concebido para apoiar automação de testes de aspectos funcionais das aplicações baseadas na web, ampla gama de plataformas e navegadores.

Funcionalidades:

» Oferece o suporte para a execução do teste paralelo que reduz o tempo gasto na execução de testes.

» Selenium necessita de recursos muito menor quando comparado a outras ferramentas de teste.

» Os casos de teste preparados usando esta ferramenta de teste pode ser executado em qualquer sistema operacional.

» Suporta as muitas linguagens de programação conhecidas como Java, Python, C #, Perl, PHP e JavaScript.

Link: http://www.seleniumhq.org/download/.

SikuliX

O SikuliX automatiza tudo o que se vê na tela do seu computador desktop executando Windows, Mac ou Linux / Unix. Ele usa o reconhecimento de imagem desenvolvido pelo OpenCV para identificar componentes da GUI. Isso é útil nos casos em que não há acesso fácil aos internos de uma GUI ou ao código-fonte do aplicativo ou página da Web em que se deseja atuar.

Link: http://sikulix.com/.

CAPÍTULO 3
Gestão de Estimativas e dos Custos

A estimativa de teste funciona como a estimativa do gerenciamento de um projeto normal, ou seja, vai aproximar o tempo que uma atividade levaria para ser concluída. E isso poderá ser estendido para o custo também.

No caso da Gestão de Custo e Estimativas de Testes, é necessário focar no esforço para o custo e prazo de uma, muitas ou as tarefas como um todo. Por isso, o mesmo questionamento que é feito em um projeto de desenvolvimento, é feito em um projeto de testes: Quanto tempo levará esses testes? Quanto vai custar? Quais recursos devemos estimar?

Portanto, para ficar claro o que é necessário estimar, temos esses grupos:

- **Recursos:** são necessários para executar quaisquer tarefas do projeto, como pessoas, equipamentos, instalações etc.
- **Prazo:** é o recurso mais valioso de um projeto e todo projeto, por definição, tem um prazo para entrega.
- **Recursos Humanos:** conhecimento e a experiência dos membros da equipe.
- **Custo:** significa o quanto, em valor monetário, é necessário para concluir o projeto.

Após a definição de alguns elementos essenciais para o início do processo de gestão, é hora de escolher algumas ferramentas disponíveis e algumas metodologias. Uma das primeiras ações é dividir as tarefas em tarefas menores e, para isso, existem várias opções e ferramentas.

Essa técnica utiliza uma (*Work Breakdown Structure*) Estrutura de Divisão de Trabalho onde, uma tarefa complexa é dividida em sub tarefas que, por sua vez, é dividida em funcionalidades. O objetivo desta atividade é criar tarefa como detalhado quanto possível. As ferramentas mais adequadas para ilustrar esse processo pode ser uma planilha ou uma WBS.

Em seguida, deve alocar as tarefas aos membros da equipe, obviamente, capaz de realizar tal função. Para essa etapa pode aproveitar a planilha ou a WBS que foram geradas na etapa anterior, ou criar um modelo 5W2H.

Sabendo o que cada membro da equipe vai fazer e tendo em vista suas habilidades, esse é o momento de estimar os esforços para as tarefas. Para isso, é possível utilizar o Ponto de Função, já estudados em módulos anteriores:

O tamanho da tarefa depende do tamanho funcional do sistema em teste, que reflete a quantidade de funcionalidades importantes para os usuários. Quanto mais funcionalidades, mais complexo é o sistema.

Os pontos funcionais, antes do início das tarefas de estimativa, são divididos em três grupos: Complexo, Médio e Simples, conforme a tabela abaixo:

Tabela 6. Estimativa de ponto de função.

Tipo	Valor
Complexo	X pontos
Médio	X menos pontos
Simples	X menos pontos

Fonte: elaboração própria do autor.

Modelagem algorítmica de custos

Segundo Sommerville (2011, p. 443), a modelagem algorítmica de custos utiliza uma fórmula matemática que prevê os custos de um *software* considerando estimativas como tamanho do projeto, tipo de software, detalhes sobre a equipe e outros fatores sobre o processo e o produto. Ao analisar os custos e atributos de projetos concluídos pode-se construir um modelo algorítmico e encontrar a fórmula que melhor represente a realidade, fazendo estimativas dos custos de desenvolvimento e testes de *softwares*.

Ao classificar a complexidade dos pontos de função, é possível estimar a duração, ou seja, quanto tempo é necessário para concluir a tarefa de testá-los.

Figura 1. Fórmula do Esforço total.

$$Esforço\ total = Total\ de\ PF * Estimativa$$

Fonte: Sommerville (2011, p. 443)

» **Esforço total:** o esforço para testar completamente todas as funções do site.

» **Total de pontos de função:** total de módulos do site.

» **Estimativa definida por pontos de função:** o esforço médio para concluir um ponto de função. Esse valor depende da produtividade do membro que se encarregará dessa tarefa.

Por exemplo, depois de todo o processo de definição de pontos de função, a equipe definiu como 3h/pf (Horas/Ponto de Função). Com uma planilha ou ferramenta própria para WBS, poderia ser apresentado algo como:

Tabela 7. Estimativas por Ponto de Função.

Tipo	Ponderação	Núm. de PF	Total
Complexo	5	12	60
Médio	3	7	21
Simples	1	3	3
Total de PF			84
Estimativa			3
Total estimado			**252 horas/pessoas**

Fonte: elaboração própria do autor.

E essas tarefas, quanto custam?

Para isso, é necessário saber a média de custo de cada membro da equipe e multiplica-lo pelos valores estimados, pode ser pelo total ou por cada tarefa. Lembrando que é importante que os valores chegam mais próximos da realidade e essa é uma função do gerente do projeto.

Técnica em Experiências Anteriores

Outra técnica de estimativa é a baseada em experiências anteriores. Essa técnica é muito utilizada em gestão de projetos e os valores estimados são produzidos baseando em experiências anteriores do gerente de projetos e o esforço real dispendido nesses projetos em atividades relacionadas ao desenvolvimento de software.

"Normalmente, se identifica os entregáveis que devem ser produzidos em um projeto e os diferentes componentes de *software* ou sistemas que serão desenvolvidos. Geralmente, é útil contar com um grupo de pessoas envolvidas na estimativa de esforço e pedir a cada membro do grupo para explicar sua estimativa. Isso muitas vezes revela fatores que outros não levaram em consideração. A dificuldade com as técnicas baseadas em experiências é que um novo projeto de *software* pode não ter muito em comum com projetos anteriores" (SOMMERVILLE 2011, p. 443).

Suas estimativas podem se basear em cenários. Por exemplo:

» o Cenário ideal;

» o Cenário provável;

» o Cenário pior.

Por exemplo, supondo as três situações:

Tabela 8. Cenários.

A	Cenário ideal	100 horas/pessoas
B	Cenário provável	150 horas/pessoas
C	Cenário Pior	200 horas/pessoas

Fonte: elaboração própria do autor.

A fórmula para o cálculo da média ponderada é a seguinte:

$$Estimativa = \frac{a + 4b + c}{6}$$

No caso do exemplo:

$$Estimativa = \frac{100 + 4 * 150 + 200}{6}$$

Segundo Sommerville (2011, p. 444), "se usar um modelo algorítmico de estimativa de custos, deve desenvolver uma série de estimativas (piores, esperadas e melhores), ao invés de uma única estimativa, e aplicar a fórmula de custo para todas elas. É mais fácil as estimativas serem precisas quando se entende o tipo de *software* que está sendo desenvolvido, tenha calibrado o modelo de custo usando dados locais, ou quando opções de hardware e linguagem de programação são predefinidas".

Todos os modelos algorítmicos têm problemas semelhantes:

1. Muitas vezes, é difícil estimar o Tamanho em um estágio de um projeto, quando apenas a especificação está disponível. Estimativas de ponto de função e ponto de aplicação (ver adiante) são mais fáceis de se produzir do que as estimativas do tamanho de código, mas, às vezes, ainda são imprecisas.
2. As estimativas dos fatores que contribuem para a e b são subjetivas. As estimativas variam de uma pessoa para outra, dependendo de sua formação e experiência com o tipo de sistema que está sendo desenvolvido.

Logo após criar as estimativas, é recomendável que seja revisada e, dada sua importância, seja aprovada por algum conselho administrativo, composto normalmente por um CEO ou Diretor.

CAPÍTULO 4
Gestão dos profissionais

Segundo Sommerville (2011, p. 424), “montar uma equipe que tenha o equilíbrio entre as habilidades técnicas, a experiência e as personalidades é uma tarefa de gerenciamento crítico. No entanto, os grupos bem-sucedidos são mais do que um conjunto de indivíduos com equilíbrio de habilidades. Um bom grupo é coeso e tem espírito de equipe. As pessoas envolvidas são motivadas pelo sucesso do grupo, bem como por seus próprios objetivos pessoais”

Segundo o PMBOK, o gerenciamento dos recursos humanos do projeto inclui os processos que organizam e gerenciam a equipe do projeto. A equipe do projeto consiste nas pessoas com papéis e responsabilidades designadas para a conclusão do projeto. O tipo e o número de membros da equipe do projeto podem mudar com frequência ao longo do projeto. Os membros da equipe do projeto também podem ser referidos como pessoal do projeto”.

O PMBOK confere ao gerente de projetos a missão de montar e organizar a equipe que irá trabalhar no projeto de testes. Uma característica importante de todos os gerentes de teste bem-sucedidos é a organização e o gerenciamento de uma equipe de teste de alto desempenho que fornece valor comercial à organização.

Segundo Pressman (2011, p. 567), o People-CMM define as seguintes práticas a gestão de pessoal:

- formação da equipe;
- comunicação;
- ambiente de trabalho;
- gerenciamento de desempenho;
- treinamento;
- compensação;
- análise de competências;
- desenvolvimento de carreira.

Desenvolver o Plano de Recursos Humanos

O planejamento de recursos humanos é um processo que identifica as necessidades atuais e futuras de recursos humanos de uma organização. O objetivo do planejamento de recursos humanos é garantir o melhor ajuste entre os membros da equipe e os projetos e evitar a falta ou redundância de mão de obra.

Segundo o PMBOK (2011, p. 218), "desenvolver o plano de recursos humanos é o processo de identificar e documentar papéis, responsabilidades, habilidades necessárias e relações hierárquicas do projeto, e criar um plano de gerenciamento de pessoal. O planejamento de recursos humanos é usado para determinar e identificar recursos humanos com as habilidades necessárias para o êxito do projeto. O plano de recursos humanos documenta papéis e responsabilidades do projeto, organogramas do projeto e o plano de gerenciamento de pessoal, incluindo o cronograma para mobilização e liberação de pessoal".

Gestor de Testes

Primeiramente é definido um Gestor de Testes. Normalmente essa pessoa já foi gerente de projetos de desenvolvimento e já possui as habilidades interpessoais necessárias para a gestão de testes. No gerenciamento da equipe também é necessário que o Gestor de Testes tenha habilidade em lidar com situações de conflitos. Quando existem essas hostilidades e elas começam a prejudicar o trabalho da equipe, o gerente de projetos deve facilitar a resolução dessas animosidades, procurando manter o ambiente favorável para o bom desempenho das funções de cada colaborador. A negociação é o meio mais utilizado para resolver esses fatos, e é uma das habilidades mais exigidas do gerente de projetos.

No mundo ideal, o gerente de projetos gostaria de escolher os componentes e melhores técnicos para o projeto, de acordo com sua percepção e gosto. Muitas vezes, há indivíduos disponíveis que dominam e conhecem o projeto e participou do projeto de desenvolvimento.

Segundo Sommerville (2011, p. 424) em relação aos bons gestores, sempre devem tentar incentivar a coesão do grupo. Eles podem organizar eventos sociais para os membros do grupo e suas famílias, além de tentar estabelecer um senso de identidade para o grupo, nomeando e estabelecendo uma identidade e território para o grupo, ou eles podem envolver-se em atividades explícitas de construção de grupos, como esportes e jogos".

Gerenciar e liderar a equipe também inclui, entre outras atividades:

» Influenciar a equipe do projeto.

» Comportamento profissional e ético.

Membros da Equipe

A equipe do projeto é um grupo de pessoas que trabalham juntos para alcançar o objetivo geral do projeto. As seguintes características são essenciais para formar uma equipe altamente eficaz.

» **Forte cooperação**

Cooperação é o ato de trabalhar com os outros e de agir em conjunto para realizar um trabalho. O DevOps e metodologias ágeis são especialistas no assunto.

» **Compromisso**

Todos os membros assumem o compromisso com os objetivos comuns do projeto, importando e assumindo a responsabilidade pelo trabalho da equipe.

» **Comunicação**

Os membros falam e expressam suas ideias de maneira clara, honesta e lógica, para que possam se entender.

» **Compartilhamento**

Em uma boa equipe, os membros estão dispostos a compartilhar informações, conhecimentos e experiências para se capacitarem.

> "Os membros de um grupo coeso pensam que o grupo é mais importante do que seus indivíduos. Os membros de um grupo coeso e bem liderado são leais ao grupo. Eles se identificam com os objetivos do grupo e com os outros membros do grupo. Eles tentam proteger o grupo das interferências externas como uma entidade. Isso torna o grupo forte e capaz de lidar com problemas e situações inesperadas" (SOMMERVILLE, p. 424).

Nem sempre será possível montar uma equipe com personalidade que se complementa. Se isso acontecer o Gestor de projeto precisa controlar o grupo para que suas metas

individuais não tenham precedência sobre os objetivos organizacionais e os do grupo. Esse controle é mais fácil de obter se todos os membros do grupo participarem de todos os estágios do projeto. A iniciativa individual é mais provável quando aos membros do grupo são dadas instruções sem que eles estejam cientes do papel que sua tarefa tem no projeto geral.

Embora os papéis e responsabilidades específicas para os membros da equipe do projeto sejam designadas, o envolvimento de todos os membros da equipe no planejamento do projeto e na tomada de decisões pode ser benéfico. O envolvimento e a participação dos membros da equipe desde o início agregam seus conhecimentos durante o processo de planejamento e fortalece o compromisso com o projeto.

Avaliação de Competências

As competências dos membros são um ponto importante que se deve considerar no planejamento de recursos, onde é necessário associar membros com diferentes competências à tarefa correta. A habilidade necessária dos membros para concluir diferentes tarefas do projeto. As habilidades e habilidades dos membros da equipe devem ser avaliadas em relação à missão e objetivo do projeto. Se as pessoas disponíveis não possuírem as competências necessárias, o Gerente de Teste deve planejar como capacitá-las.

As competências dessa equipe se referem às possibilidades de indivíduos trabalharem em conjunto, com o intuito de atingirem o objetivo do projeto. Para conseguir essa coesão de objetivos de tantas pessoas, devem ser feitas: reuniões e encontros em espaços físicos específicos para uso do time; divulgações dos resultados do grupo; e estruturações da equipe quanto às responsabilidades de cada componente.

Treinamento e Avaliação

O planejamento de Recursos Humanos incluirá uma consideração de como os membros existentes podem ser treinados e desenvolvidos para atingir as habilidades e competências necessárias. O plano de treinamento deve ser criado e aplicado logo após a identificação da lacuna. É responsabilidade do Gerente de Teste identificar quais habilidades os membros não possuem para criar um plano de treinamento apropriado para eles. Os programas de treinamento são monitorados e avaliados com frequência para garantir que sejam eficazes. Este programa pode ser alterado se necessário.

Ferramentas

Existem diversos formatos para documentar os papéis e responsabilidades dos membros da equipe. A maioria dos formatos correspondem a um de três tipos:

» **Gráficos hierárquicos**

A estrutura de organograma tradicional pode ser usada para mostrar posições e relações em um formato gráfico de cima para baixo.

» **Gráficos matriciais**

Uma matriz de responsabilidades (MR) é usada para ilustrar as conexões entre pacotes de trabalho ou atividades e os membros da equipe do projeto.

» **Em formatos de texto**

As responsabilidades de membros da equipe que requerem descrições detalhadas podem ser especificadas em formatos de texto. Normalmente em forma de uma lista organizada ou formulário, esses documentos fornecem informações como responsabilidades, autoridade, competências e qualificações.

DOCUMENTAÇÃO

UNIDADE IV

CAPÍTULO 1

Conceito e Necessidade de documentação

Existem alguns princípios básicos e fundamentais para que a documentação de um *software* seja bem elaborada. Muitos desenvolvedores acabam por considerar somente o código como parte essencial de um sistema, porém para que o produto verdadeiramente seja efetivo e solucione realmente o problema de alguém, ele precisa estar devidamente documentado.

Não importa quanto trabalho e dedicação o desenvolvedor coloque em seu código, se o *software* chegar ao cliente sem documentação ou com uma documentação que foi escrita sem a devida importância e atenção às melhores práticas, os usuários jamais sentir-se-ão à vontade e seguros em sua utilização e com o passar do tempo tenderão a escolher outro sistema bem documentado e de fácil entendimento, conhecido no termo em inglês como "*user-friendly*".

Mesmo que o seu *software* seja o melhor naquilo que ele se propõe a solucionar, ele não servirá ao seu propósito se for o único que conseguir entendê-lo. Exemplos práticos e tutorias bem detalhados devem fazer parte de sua documentação, além de orientações de como realizar conexões com APIs entre outras instruções.

Além disso, a documentação não é importante somente para os usuários do sistema, mas para o próprio desenvolvedor, que no futuro pode precisar alterar ou aprimorar códigos e verificar como foram criados as lógicas e o funcionamento do sistema, como também utilizar esta documentação como base para instruir novos desenvolvedores na contínua manutenção e atualização da ferramenta. Mesmo que o desenvolvedor tenha o hábito de comentar seus códigos, a documentação é aliada quando depois de algum tempo do desenvolvimento é preciso mais esclarecimentos e detalhes sobre as notas deixadas no código.

Escrever uma documentação precisa para o seu sistema dá suporte para o crescimento dele a formação de uma comunidade de usuários que interagem e mantém o sistema vivo, ganhando maturidade e sucesso ao longo do tempo.

No momento em que se escreve a documentação de um produto, é preciso ter em mente para quem está escrevendo, para que tipo de público está escrevendo e qual o formato e linguagem adequados para que o conteúdo da documentação faça sentido para esse público em específico.

Uma documentação que de fato contribua para uma boa experiência do usuário deve conter:

1. **Tutoriais**

   Fornecem uma visão clara aos usuários sobre o que o sistema pode entregar a eles. Mostram aos usuários como completar uma tarefa utilizando o sistema os guiando por uma série de passos em ordem que completam uma ação. São desenvolvidos para usuários iniciantes, explicando minuciosamente cada interação necessária com o sistema.

2. **Guias de Referências Técnicas**

   Refere-se a informações relevantes sobre o código do seu sistema.

   Oferece uma descrição básica de como utilizar classes e métodos do seu *software* e é a parte da documentação que os desenvolvedores têm mais facilidade de escrever, pois do código e suas especificações eles entendem bem.

É importante garantir que a sua documentação seja facilmente encontrada e de acesso rápido aos usuários. Também é muito importante que sua documentação seja apresentada em uma estrutura que conte com uma ferramenta de pesquisa. Assim o usuário pode consultar e encontrar mais facilmente orientações sobre determinadas partes e funcionalidades do sistema, sem a necessidade de percorrer o documento inteiro, podendo priorizar demandas.

Ainda é necessário disponibilizar uma seção de Dúvidas Frequentes, que é estruturada considerando as principais perguntas recebidas de seus usuários.

Garanta também que sua documentação seja sempre atualizada, sempre que houver qualquer mudança e alteração no produto. Somente assim ela será uma fonte confiável e relevante para os usuários. Quando houver atualizações, novidades

e outras informações sobre o sistema, expanda seus canais de comunicação para além da documentação.

É possível utilizar posts em um blog dentro do site do sistema, por exemplo, para explicar a atuais e potenciais usuários como determinadas partes do sistema funcionam, as últimas inovações e atualizações, dicas, tutoriais e até estabelecer um fórum sobre o produto.

Sem dúvida alguma, uma documentação bem estruturada aliada a canais de comunicação que evidenciem a importância de suas informações é fundamental para o crescimento de qualquer sistema.

Como principal retorno pelo tempo e conhecimento investidos em uma documentação bem escrita, o desenvolvedor ganha mais agilidade na implantação do *software* e maior credibilidade perante seus usuários, o que aumenta seu faturamento e sua sobrevivência no mercado.

CAPÍTULO 2

ADL (Architecture Description Language)

Conforme um sistema ganha proporções significativas, torna-se indispensável a utilização de métodos para especificar a arquitetura de grandes *softwares* evitando ambiguidades em sua linguagem de programação.

As ADLs – *Architecture Descripition Language*, representam a arquitetura de um software, ou seja, os componentes que compõe o sistema e os padrões e mecanismos de interação entre eles. Como componentes consideramos módulos, tarefas, funções etc. A arquitetura de um *software* demonstra o tipo de componente mais apropriado para visualização ou ilustrar componentes diferentes em uma mesma visualização.

As arquiteturas básicas, em sua representação informal, normalmente são representadas por desenhos que muitas vezes não deixam claro a natureza dos componentes, suas propriedades, conexões e como se refletem nas funcionalidades do sistema como um todo.

Essas imagens pretendem demonstrar um cenário intuitivo da construção do sistema, mas não respondem a perguntas básicas como:

Quais são os componentes? Os módulos são compilados em que momento? As tarefas e processos são encadeados a partir de módulos diferentes? O que os componentes fazem e com quais outros componentes eles se relacionam? O que significam as conexões e quais são suas combinações? Quais são os mecanismos que ligam essas conexões entre si?

Por isso surgem as ADLs, para resolver essas lacunas de informações. Elas resultam de uma abordagem linguística da representação formal das arquiteturas, solucionando as deficiências das representações informais. ADLs sofisticadas permitem análises e testes de viabilidade que facilitam diversas decisões.

Alguns exemplos atuais de ADLs são: Rapide, Unicon, ArTek, Wright e Meta-H. Uma arquitetura desempenha funções importantes no desenvolvimento de um projeto, podendo ser consultada e tratada como uma rota de autoridade, que pode ser aplicada como um ativo também para outros projetos.

As ADLS representam:

1. Base de Comunicação entre os membros da equipe e com os clientes que utilizam as ADLs para entender sobre o desenvolvimento do sistema e como ele funciona.

2. Modelo a ser seguido que auxilia nas atribuições de trabalho, escolha dos componentes, unidades de gerenciamento, cronogramas e estrutura de detalhamento do trabalho, além de facilitar os planos de integração, testes e processos de manutenção.

3. Projeto para desenvolvimento de uma linha inteira de produtos, pois uma vez definida a arquitetura, ela pode ser reutilizada em outros sistemas. Com uma boa gestão, uma família inteira de produtos pode ser produzida utilizando uma mesma arquitetura e nesse sentido, a importância da arquitetura é ampliada.

4. As primeiras decisões de um projeto, servindo como mapa de requisitos e seleção de componentes e conexões.

5. Consequências de grande proporção e impacto em um sistema quando há necessidade de promover mudanças.

6. As primeiras oportunidades de avaliar o desempenho, complexidade e volume de comunicação e coordenação de componentes e o que essas escolhas e decisões representam para o nível de qualidade de um projeto.

Algumas arquiteturas oferecem a possibilidade de análises automatizadas e outras acompanham estratégias de avaliação que podem ser aplicadas à arquitetura. Assim, decisões e atributos de qualidade podem ser testados antes que não possam ser mais alterados.

Existem algumas propriedades importantes que as ADLs devem apresentar. São elas:

1. Capacidade de representar componentes juntamente com propriedades, interfaces e implementações.

2. Capacidade de representar conectores com protocolos.

3. Abstração e Encapsulamento.

4. Capacidade de verificar tipos e suportar ferramentas de análise.

5. Promover a integração da comunicação de componentes conectados entre si.
6. Capacidade de modelar arquiteturas dinâmicas e administrar causalidade e tempo.
7. Suportar o refinamento hierárquico.
8. Mapear comportamentos para arquiteturas para verificação de conformidades.

ADLS e outras linguagens

ADL e UML

A Linguagem de Descrição da Arquitetura (ADL) tem como característica a definição de ser uma linguagem flexível, ou seja, suporta linguagens gráficas, textuais ou ambas, destinadas a escrever um sistema de software, baseando-se em seus elementos arquitetônicos e seus relacionamentos.

Em outras palavras, ADL é uma linguagem que permite formalização, descrição, especificação, modelagem e raciocínio em arquiteturas de software. Cada um desses recursos deve ser cumprido por uma linguagem que é proclamada como ADL. Uma boa ADL deve fornece abstrações adequadas para modelar um sistema grande. Cada ADL incorpora uma abordagem específica para a especificação e evolução da arquitetura.

Análise de Domínio Orientado a Recursos

A análise de recursos é uma ferramenta para certos métodos de análise de domínio, como o método *Feature-Oriented Domain Analysis*; ele prossegue catalogando os recursos do sistema visíveis ao usuário de maneira estruturada.

Os recursos são estruturados nas três categorias a seguir: recursos orientados ao sistema, recursos orientados a linguagem e recursos orientados a processos.

Recursos orientados ao sistema

Os recursos orientados ao sistema estão relacionados ao sistema de aplicativos derivado da descrição da arquitetura. Por exemplo, certas ADLs podem não ser capazes de expressar restrições em tempo real sobre os componentes arquiteturais de um sistema,

enquanto outras podem. Todos os recursos desta categoria são atributos de um sistema final; no entanto, eles refletem sobre a capacidade da ADL de expressar ou descrever esses atributos no nível da arquitetura.

Recursos orientados a idiomas

Recursos orientados a linguagem são recursos da própria ADL, independentemente do (s) sistema (s) que está sendo usado para desenvolver. Esses atributos incluem o tipo de informação normalmente encontrada em um manual de referência de idioma. Um exemplo é como a sintaxe e a semântica da ADL são formalmente especificadas e quais abstrações arquitetônicas a ADL incorpora.

Recursos orientados ao processo

Recursos orientados a processos são recursos de um processo relacionado ao uso da ADL para criar, validar, analisar e refinar uma descrição da arquitetura e criar um sistema de aplicativos a partir dela. Os componentes incluídos aqui são os atributos que medem ou descrevem como ou até que ponto uma ADL permite a avaliação preditiva do sistema de aplicativos com base nas informações no nível da arquitetura. Esses atributos medem se a ADL contém ou não informações suficientes para tornar-se uma arquitetura analisável, independentemente da existência ou não de ferramentas que exploram esse recurso.

CAPÍTULO 3
Padrões de Documentação

Sabemos que todos os produtos de desenvolvimento de *softwares* devem ser documentados, sejam eles resultados de equipes pequenas ou de grandes corporações. Existem diversos tipos de documentos que podem ser criados, seja para explicar as funcionalidades do produto, seja para reunir informações sobre o projeto e permitir a discussão entre as partes interessadas e os desenvolvedores.

É preciso prestar muita atenção à qualidade da documentação, visto que erros podem representar lacunas entre as visões das partes interessadas e dos engenheiros, o que pode fazer com que a solução apresentada não corresponda às expectativas iniciais.

Definir o tipo de documentação depende da escolha da abordagem de desenvolvimento de *software* escolhida. Existem dois tipos de abordagem:

1. **Abordagem Waterfall (Cascata)**

   É um método com objetivos distintos para cada fase de desenvolvimento. Com essa abordagem, as equipem levam um tempo significativo para planejar o produto no início do projeto. Antes que o desenvolvimento comece, a equipe se esforça para criar toda a documentação de forma extremamente detalhada. Esta abordagem é viável para projetos que irão ter pouca ou nenhuma necessidade de alteração em andamento, o que permite orçamento e prazos bem previsíveis. É bastante ineficaz para projetos de longo prazo.

2. **Abordagem Agile (Ágil)**

   De acordo com a 9ª Pesquisa Global de Gerenciamento de Projetos do PMI, a abordagem Agile é usada por 71% das empresas em seus projetos. Esta abordagem é baseada em um trabalho em equipe extremamente integrada com relação muito próxima de colaboração entre a equipe, os clientes e as partes interessadas.

   Promove a flexibilidade e facilita respostas ágeis às mudanças, como o próprio nome sugere. O desenvolvimento é definido por blocos de interações, que incluem planejamento, análise, design, desenvolvimento e testes.

A Abordagem Agile não exige documentação detalhada no início do projeto, pois pode haver mudanças conforme o projeto evolui. Como é a abordagem mais utilizada, apresentaremos a seguir os tipos de documentação referentes à esta abordagem.

Tipos de Documentação

Existem diversos tipos de documentação que garantem desenvolvedores e partes interessadas integrados e na mesma direção para atingir os objetivos de um projeto.

A documentação de *software* pode ser dividida em duas categorias: **Documentação do Produto e Documentação do Processo**.

A documentação do produto é uma descrição do produto que está sendo desenvolvido com instruções sobre como realizar diversas tarefas ao utilizá-lo. É dividida em documentação do sistema e documentação do usuário. A documentação do sistema descreve suas partes incluindo requisitos, decisões sobre design e arquitetura, código fonte e guias de ajuda. Já a documentação do usuário apresenta manuais especialmente elaborados para entendimento dos usuários finais sobre como utilizar todos os recursos do sistema. São tutoriais e passos para solução de problemas, além de instruções para sua instalação.

Já a documentação do processo reúne todos os documentos produzidos durante o desenvolvimento do sistema. São planejamentos, cronogramas, relatórios de testes, atas de reuniões etc. Como o próprio nome diz, a documentação do processo relata o processo de desenvolvimento enquanto a documentação do produto descreve o produto que está em desenvolvimento.

O documento de requisitos faz parte da documentação do produto. É de extrema importância, pois nele estão especificadas de forma concisa informações sobre as funcionalidades, recursos e regras de negócio do sistema, o que ele deve fazer.

No início do documento de requisitos, deve-se apresentar os papéis e responsabilidades dos participantes do projeto, quem é o proprietário do projeto e quem são as partes interessadas. Também devem constar os objetivos da equipe e os objetivos de negócio.

Além disso, é preciso deixar claro o objetivo estratégico do projeto:

1. Por que esse produto está sendo desenvolvido?
2. Como as ações da equipe afetam o desenvolvimento e cumprem os objetivos do negócio?

Ainda é importante definir premissas, uma lista de orientações técnicas ou comerciais que podem ser importantes para a equipe, considerando as ações e resultados que o negócio pretende alcançar.

Conforme o projeto evolui, surgem perguntas que podem ser registradas para resolução. Também surgem ideias de coisas que não estão sendo providenciadas no momento, mas serão feitas em um futuro breve. Faça uma lista dessas ideias para posterior consulta e organização do trabalho em equipe para priorizar os recursos.

Também pode-se acrescentar links e âncoras para tornar as informações mais abrangentes facilitando a leitura e pesquisa no documento. Se for um documento extenso, use recursos de diagramação e ilustrações para torná-lo leve e de fácil leitura. Uma boa dica é utilizar diagramas de processos de requisitos.

Documento de Arquitetura de *Software*

Incluem as principais decisões de arquitetura de *software*, não todas, mas igualmente relevantes. Deve conter um modelo de documento de design, que apresenta um consenso com as partes interessadas sobre um modelo para mapear as soluções arquitetônicas. Evidencie a arquitetura de orientação e os princípios de design que serão utilizados para produzir o produto.

Crie também uma seção de descrição da história do usuário, para que se conecte essas experiências dos usuários com os processos e cenários associados ao sistema.

Outra seção importante chama-se detalhes da solução. Nela lista-se os serviços contemplados, módulos e componentes. Uma representação esquemática da solução pode auxiliar no entendimento e na comunicação dos princípios de estrutura e design.

Documento do Código-Fonte

Constitui uma seção técnica que explica como o código funciona para os engenheiros de software. Apresenta entre outros detalhes, estrutura de geração de HTML, tipo de ligação de dados, padrões de design, medidas de segurança, entre outras informações referentes ao código.

Documento de Garantia de Qualidade

Deve conter estratégias de teste, planos de testes, especificações de casos de testes, listas de verificação, descrevendo a abordagem apropriada de testes para o software, com

detalhes sobre a estrutura da equipe, recursos necessários e o que deve ser priorizado durante os testes. Prazos, funções, responsabilidades, métodos de testes, descrição das ações de verificação de funcionalidades do produto também devem constar neste documento que é geralmente produzido por uma equipe de controle de qualidade.

Uma lista de verificação de testes facilita o acompanhamento dos testes que devem ser realizados em momentos determinados, quantos foram concluídos, quantos falharam etc.

Guia de Manutenção e Ajuda

Deve descrever problemas conhecidos com o sistema e suas soluções. Sobre a manutenção, deve apresentar as dependências entre as diferentes partes do sistema.

Documentação do Usuário

Deve ser estruturada de acordo com as diferentes tarefas dos usuários e os diferentes níveis de suas experiências com o software. Deve ser explicada da forma mais curta possível, evidenciando como o *software* pode resolver os problemas desses usuários finais.

Tutorias, treinamentos de integração, perguntas frequentes, portais de suporte são aliados de uma documentação consistente que agregue valor para os usuários. Coletar os comentários dos clientes constantemente também ajuda a manter sua documentação atualizada.

Para administradores de sistemas, o documento não precisa conter informações sobre como operar o software, mas sim como instalá-lo e ter acesso às atualizações que irão ajudá-lo na manutenção efetiva do sistema. Para eles, é importante a descrição das funcionalidades do *software* e um guia de administração que aborde os comportamentos esperados do sistema em ambientes diversos e em casos de integração com outros sistemas, além de informações sobre como proceder em caso de mau funcionamento do software.

CAPÍTULO 4
Visão Arquitetural

Um documento de arquitetura de *software* é um mapa do software. Usamos para ver de relance como o *software* está estruturado. Isso ajuda a entender os módulos e componentes do *software* sem se aprofundar no código. É uma ferramenta para se comunicar com outras pessoas – desenvolvedores e não desenvolvedores – sobre o software.

As visualizações de arquitetura são representações da arquitetura geral que são significativas para uma ou mais partes interessadas no sistema. O arquiteto escolhe e desenvolve um conjunto de visualizações que permitirá que a arquitetura seja comunicada e compreendida por todas as partes interessadas e que verifique se o sistema abordará suas preocupações.

Uma arquitetura é geralmente representada por meio de um ou mais modelos de arquitetura que juntos fornecem uma descrição coerente da arquitetura do sistema. Um modelo único e abrangente geralmente é complexo demais para ser entendido e comunicado em sua forma mais detalhada, mostrando todos os relacionamentos entre os vários componentes comerciais e técnicos. Assim como na arquitetura de um edifício, normalmente é necessário desenvolver várias visões da arquitetura de um sistema de informações, para permitir que a arquitetura seja comunicada e compreendida pelas diferentes partes interessadas no sistema.

Segundo ISO/IEC/IEEE 42010:2011, "o conceito da arquitetura de um sistema, conforme expressa em uma descrição da arquitetura, auxilia o entendimento da essência do sistema e das principais propriedades pertencentes ao seu comportamento, composição e evolução, que por sua vez afetam preocupações como a viabilidade, utilidade e manutenção do sistema".

As descrições de arquitetura são usadas pelas partes que criam, utilizam e gerenciam sistemas modernos para melhorar a comunicação e a cooperação, permitindo que trabalhem de maneira coerente e integrada. Estruturas de arquitetura e linguagens de descrição de arquitetura estão sendo criadas como ativos que codificam as convenções e práticas comuns de arquitetura e a descrição de arquiteturas em diferentes comunidades e domínios de aplicativo.

Por exemplo, uma arquitetura de aplicativo de três camadas se parece com isso:

Por que precisamos de documentação de software?

Antes de descrever como criar a documentação arquitetural corretamente, precisamos entender por que ela é necessária. Existem três objetivos principais para a documentação arquitetural:

» **Compartilhamento de conhecimento**

É adequado para transferência de conhecimento entre pessoas que trabalham em diferentes áreas funcionais do projeto, bem como para transferência de conhecimento para novos participantes.

» **Comunicação.**

A documentação é o ponto de partida para a interação entre diferentes partes interessadas. Em particular, ajuda a compartilhar as ideias do arquiteto com os desenvolvedores.

» **Análises.**

A documentação também é um ponto de partida para futuras revisões arquitetônicas do projeto.

Os seguintes conceitos são centrais para o tópico de visualizações. Esses conceitos foram adaptados de definições mais formais contidas na ANSI / IEEE Std 1471-2000:

» **Sistema** é uma coleção de componentes organizados para realizar uma função específica ou um conjunto de funções.

» **Arquitetura de um sistema** é a organização fundamental do sistema, incorporada em seus componentes, seus relacionamentos entre si e com o meio ambiente e os princípios que orientam seu design e evolução.

» **Uma descrição da arquitetura** é uma coleção de artefatos que documentam uma arquitetura.

» **As partes interessadas** são pessoas que têm papéis-chave ou preocupações sobre o sistema; por exemplo, como usuários, desenvolvedores ou gerentes.

» **As preocupações** são os principais interesses que são de importância crucial para as partes interessadas no sistema e determinam a aceitabilidade do sistema.

» **Uma visão** é uma representação de um sistema inteiro da perspectiva de um conjunto de preocupações relacionadas.

Ao capturar ou representar o design de uma arquitetura de sistema, o arquiteto normalmente cria um ou mais modelos de arquitetura, possivelmente usando ferramentas diferentes. Uma visão compreenderá partes selecionadas de um ou mais modelos, escolhidas para demonstrar a uma parte interessada específica ou grupo de partes interessadas que suas preocupações estão sendo adequadamente tratadas no design da arquitetura do sistema.

Um ponto de vista define a perspectiva da qual uma visão é obtida. Mais especificamente, um ponto de vista define: como construir e usar uma visão (por meio de um esquema ou modelo apropriado); as informações que devem aparecer na exibição; as técnicas de modelagem para expressar e analisar as informações; e uma justificativa para essas escolhas (por exemplo, descrevendo o objetivo e o público-alvo da visão).

» UML.

» Modelo de vista arquitetônica 4 + 1.

» Registros de Decisão de Arquitetura.

» O modelo C4.

» Diagramas de dependência.

» Mapa do aplicativo.

UML

Um dos modelos arquiteturais é a UML, pois, ela possui todos os elementos necessários para a demonstração do sistema. Ela combina conceitos de várias metodologias, abrange modelagem de negócios, requisitos, análise, desenho, implementação, testes e implantação. Independente de linguagem, plataforma ou processo, ela suporta a aplicação em vários domínios.

Atualmente, há muito interesse em usar a UML para descrição arquitetônica: as técnicas pelas quais os arquitetos esboçam, capturam, modelam, documentam e analisar o conhecimento de arquitetura e as decisões sobre sistemas que usam muito software.

Essas técnicas permitem que os arquitetos registrem o que estão fazendo, modifiquem ou manipulem as arquiteturas candidatas, reutilizem partes das arquiteturas existentes e comunicar informações arquitetônicas a outras pessoas. Essas descrições podem ser usadas para analisar e raciocinar sobre a arquitetura – possivelmente com automação Apoio, suporte. As análises variam de avaliar a viabilidade.

IEEE P1471

A prática recomendada para descrição arquitetônica (IEEE P1471) representa um consenso emergente para a descrição das arquiteturas de sistemas intensivos de software. IEEE P1471 é a prática recomendada preliminar para a descrição arquitetônica. Foi desenvolvido pelo Grupo de Trabalho de Arquitetura do IEEE, fundado e patrocinado pelo Comitê de Padrões de Engenharia de *Software* da IEEE Computer Society. O projeto de Prática Recomendada foi produzido entre 1995 e 1998 por um grupo de aproximadamente trinta participantes e mais de 140 revisores.

O IEEE P1471 é uma prática recomendada - que é um tipo de padrão IEEE.2

Os ingredientes importantes do IEEE P1471 são:

1. Um conjunto normativo de definições para termos, incluindo descrição arquitetônica, visão arquitetônica, ponto de vista arquitetônico.

2. Um quadro conceitual que estabelece esses termos no contexto dos muitos usos de descrições arquitetônicas para construção, análise e construção de sistemas evolução do sistema.

3. Um conjunto de requisitos em uma descrição arquitetônica de um sistema.

Modelo de vista arquitetônica 4 + 1

O modelo de visualização da arquitetura 4 + 1 foi criado por Philippe Kruchten e publicado, em 1995, em seu artigo intitulado “Projetos arquitetônicos - O modelo de visualização da arquitetura de *software* “ 4 + 1 ”.

Essa maneira de visualizar uma arquitetura de aplicativo de *software* é baseada em cinco visualizações/perspectivas do aplicativo, informando quais diagramas podem ser usados para documentar cada uma dessas visualizações.

- **Visualização Lógica / Estrutural: p**reocupa-se com a funcionalidade fornecida pelo sistema e como o código é projetado para fornecer essa funcionalidade.
- **Visualização Implementação/Desenvolvedor: r**etrata a organização estática do código, os componentes, módulos e pacotes.
- **Visão Process/Behavior:** concentra-se no comportamento em tempo de execução do sistema, como os processos do sistema se comunicam, simultaneidade, sincronização, desempenho e assim por diante.
- **Implantação/visualização física:** Ilustra a organização física do aplicativo, é sobre "qual código é executado em que hardware".
- **Visualização Caso de Uso/Cenário:** a arquitetura como um todo é explicada com a ajuda de alguns casos de uso, que são simplesmente sequências de interações. Parte da arquitetura evolui desses casos de uso.

Registros de Decisão de Arquitetura

Um ADR é uma entrada de log sobre as decisões de arquitetura que foram tomadas e que levam ao estado da arquitetura como é agora ou como pretende ser no futuro. Eles contêm o porquê dos diagramas que descrevem a arquitetura.

- **Requisito de arquitetura significativa (ASR):** um requisito que tem um efeito mensurável na arquitetura de um sistema de software.
- **Decisão de arquitetura (AD):** uma opção de design de *software* que atende a um requisito significativo.
- **Registro de Decisão de Arquitetura (ADR):** um documento que captura uma importante decisão arquitetônica tomada juntamente com seu contexto e consequências.
- **Log de Decisão de Arquitetura (ADL):** a coleção de todos os ADRs criados e mantidos para um projeto (ou organização) específico.
- **Gerenciamento de conhecimento de arquitetura (AKM):** a esfera superior de todos os conceitos anteriores.

O modelo C4

O modelo C4 foi introduzido por Simon Brown, e é a melhor ideia sobre documentação de arquitetura de *software* que me deparei até agora. Explicarei rapidamente a ideia principal com minhas próprias palavras, embora usando seus próprios diagramas de exemplo.

A ideia é usar quatro níveis diferentes de granularidade (ou zoom) para documentar a arquitetura do software:

- Nível 1: diagrama de contexto do sistema.
- Nível 2: Diagrama do contêiner.
- Nível 3: diagrama de componentes.
- Nível 4: diagrama de código.

Diagramas de dependência

Os diagramas de dependência são úteis para nos informar sobre as dependências existentes nos diferentes tipos de código em nossa base de código. O mais importante aqui é que esses diagramas sejam gerados automaticamente diretamente do código; caso contrário, o diagrama refletirá apenas a aparência do código e, se isso fosse preciso, não precisaríamos muito dessa documentação.

Mapa do aplicativo

O Mapa do Aplicativo tem como objetivo ser realmente um mapa do aplicativo, definindo suas “cidades” (Componentes), suas “estradas locais” (casos de uso), “rodovias” (eventos) etc. A diferença entre os módulos e componentes é que um módulo é qualquer peça modular para a aplicação, enquanto um componente é um sábio domínio módulo da aplicação. Portanto, embora um ORM seja um módulo do aplicativo, ele não é um componente, pois lida apenas com preocupações técnicas. Por outro lado, um módulo “Faturamento” é um componente porque lida com preocupações de domínio.

A decomposição do processo

A arquitetura do processo leva em consideração alguns requisitos não funcionais, como desempenho e desempenho. Ele aborda questões de concorrência e distribuição, de

integridade do sistema, de tolerância a falhas e como as principais abstrações da visão lógica se encaixam na arquitetura do processo - em qual encadeamento de controle é uma operação para um objeto realmente executado.

ISO / IEC / IEEE 42010: 2011 - Descrição da arquitetura

Âmbito

Esta Norma especifica a maneira pela qual as descrições de arquitetura dos sistemas são organizadas e expressas, especifica pontos de vista de arquitetura, estruturas de arquitetura e linguagens de descrição de arquitetura para uso em descrições de arquitetura, também fornece motivações para os termos e conceitos utilizados; apresenta orientações sobre a especificação de pontos de vista da arquitetura; e demonstra o uso desta norma internacional com outras normas.

Termos e definições

Arquitetura

Processo de concepção, definição, expressão, documentação, comunicação, certificação da implementação adequada, manutenção e aprimoramento de uma arquitetura ao longo do ciclo de vida de um sistema. São conceitos ou propriedades fundamentais de um sistema em seu ambiente incorporado em seus elementos, relacionamentos e nos princípios de seu design e evolução.

A arquitetura ocorre no contexto de uma organização (“pessoa ou grupo de pessoas e instalações com um arranjo de responsabilidades, autoridades e relacionamentos”) e/ou projeto (“empreendimento com critérios definidos de início e término realizados” criar um produto ou serviço de acordo com os recursos e requisitos especificados”)

Descrição da arquitetura

Produto de trabalho usado para expressar uma arquitetura.

Estrutura de arquitetura

Convenções, princípios e práticas para a descrição de arquiteturas estabelecidas em um domínio específico de aplicação e/ou comunidade de partes interessadas.

Exemplos:

» Arquitetura e Metodologias de Referência Corporativa Generalizada (GERAM) [ISO 15704] é uma estrutura de arquitetura.

» O Modelo de Referência de Processamento Distribuído Aberto (RM-ODP) [ISO/IEC 10746] é uma estrutura de arquitetura.

Vista arquitetura

Produto de trabalho que expressa a arquitetura de um sistema da perspectiva de preocupações específicas do sistema

Ponto de vista da arquitetura

Produto de trabalho que estabelece as convenções para a construção, interpretação e uso de vistas da arquitetura para enquadrar preocupações específicas do sistema

Preocupação

Uma preocupação refere-se a qualquer influência em um sistema em seu ambiente, incluindo influências de desenvolvimento, tecnológicas, comerciais, operacionais, organizacionais, políticas, econômicas, legais, regulatórias, ecológicas e sociais. Interesse em um sistema relevante para um ou mais de seus stakeholders.

Ambiente

Contexto que determina a configuração e as circunstâncias de todas as influências sobre um sistema. O ambiente de um sistema inclui influências de desenvolvimento, tecnológicas, comerciais, operacionais, organizacionais, políticas, econômicas, legais, regulatórias, ecológicas e sociais.

Tipo de modelo

Convenções para um tipo de modelagem.

Exemplos de tipos de modelo incluem diagramas de fluxo de dados, diagramas de classes, redes de Petri, balanços, organogramas e modelos de transição de estados.

Parte interessada

Indivíduo, equipe, organização ou classes dos mesmos, com interesse em um sistema.

Referências

CARDOSO, A. **As abordagens da qualidade em software:** QA e QC. Developers Magazine: janeiro de 2004.

HORCH, John W. **Practical Guide to Software Quality Management**. 2a ed., Artech House Computing Library, Norwood, United States, 308 p. 2003.

PIRSIG, Robert M. **Zen e a arte de manutenção em motocicletas**. 3a ed. Editora WMF Martins Fontes, 450 p. 2015.

PRESSMAN, Roger S. **Engenharia de Software, uma abordagem profissional**. 7ª ed., AMGH Editora LTDA, Porto Alegre RS, 2011 .

ROSA, Chrystian C. **Controle e Garantia de Qualidade no Desenvolvimento de Software;** 2018, Disponível em: https://medium.com/beelabsolutions/controle-garantia-de-qualidade-no-desenvolvimento-de-software-2483a9c9ed3.

MAAYAN, Gilad D. **What Software Quality (Really) Is and the Metrics You Can Use to Measure It (2017)**. Disponível em: https://www.altexsoft.com/blog/engineering/what-software-quality-really-is-and-the-metrics-you-can-use-to-measure-it/. Acessado em 1/1/2020.

ISO/IEC 9126-1: 2003. **Engenharia de software - Qualidade de produto**. Parte 1: Modelo de qualidade.

ISO/IEC 42010: 2011 **Engenharia de sistemas e software - Descrição da arquitetura.**

MELO, W.; SHULL, F.; TRAVASSOS, G. H. **Software Review Guidelines**. Systems Engineering and Computer Science Departament. 2001. COPPE/UFRJ.

CAMPOS, V. F.; **TQC** - CONTROLE DA QUALIDADE TOTAL (No estilo japonês), Belo Horizonte/MG: 8ª edição-EDG ,1940.

GUERRA, A. C.; COLOMBO, R. M. T. **Tecnologia da Informação:** Qualidade de Produto de Software", PBQP Software, 2009.

KRUCHTEN P. **Architectural Blueprints**—The "4+1" View Model of Software Architecture.Paper published in IEEE Software 12 November 1995, pp. 42-50 https://

www.win.tue.nl/~wstomv/edu/2ip30/references/Kruchten4+1.pdf. Acessado em: 18 janeiro 2020.

NBR ISO 9000-3. **Normas de gestão da qualidade e garantida da qualidade** – **Parte 3:** Diretrizes para a aplicação da NBR 19001 ao desenvolvimento, fornecimento e manutenção de software. ABNT, 1990.

NBR ISO 9001 Sistemas da qualidade – Modelo para a garantia da qualidade em projeto, desenvolvimento, produção, instalação e serviços associados ABNT, 1994 MUNNS, A. K. & BJEIRMI, B. F. The role of project management in achieving project success. IN: **International Journal of Projetc Management** vol 14 no. 2 pp. 81-87, 1997.

PAULK, Marc C. et all. **The Capability Maturity Model:** Guidelines for Improving the Software Process Addison-Wesley, 1994.

SOMMERVILLE, **Ian Engenharia de Software/Ian Sommerville**: tradução Ivan Bosnic e Kalinka G. de O. Gonçalves; revisão técnica Kechi Hirama. — 9. ed. — São Paulo: Pearson Prentice Hall, p. 551. 2011.

Sites

https://www.sealights.io/. Acesso em 11 janeiro 2020.

https://www.it-cisq.org/. Acesso em 11 janeiro 2020.

https://testlio.com/blog/regression-testing-automated. Acesso em 11 janeiro 2020.

https://economictimes.indiatimes.com/definition/regression-testing. Acesso em: 11 janeiro 2020.

https://www.browserstack.com/guide/regression-testing> Acessado em 11/01/2020

https://www.stickyminds.com/article/get-smart-about-your-regression-tests-value. Acesso em: 11 janeiro 2020.

https://www.stickyminds.com/article/managing-acceptance-criteria-plans. Acesso em: 11 janeiro 2020.

https://www.softwaretestinghelp.com/what-is-acceptance-testing/. Acesso em: 11 janeiro 2020.

https://www.agilealliance.org/glossary/acceptance/. Acesso em: 11 janeiro 2020.

https://www.guru99.com/test-management-phases-a-complete-guide-for-testing-project.html. Acesso em 11 janeiro 2020.

https://www.thedigitalmentor.com/what-is-testing-in-sdlc/. Acesso em 11 janeiro 2020.

https://www.guru99.com/levels-of-testing.html. Acesso em 11 janeiro 2020.

https://www.seguetech.com/the-four-levels-of-software-testing. Acesso em 11 janeiro 2020.

https://www.testim.io/blog/software-testing-life-cycle/. Acesso em 11 janeiro 2020.

https://www.stickyminds.com/article/introduction-web-optimization-testing. Acesso em 11 janeiro 2020.

https://www.guru99.com/introduction-to-test-management-for-curious-manager.html. Acesso em 11 janeiro 2020.

https://www.guru99.com/how-to-organize-a-test-team.html. Acesso em 11 janeiro 2020.

https://www.sitepoint.com/writing-software-documentation/. Acesso em 11 janeiro 2020.

https://www.stickyminds.com/article/comprehensive-documentation-has-its-place> Acesso em 11 janeiro 2020.

https://www.stickyminds.com/article/tests-documentation. Acesso em 11 janeiro 2020.

https://www.todaysoftmag.com/article/2241/architecture-description-languages. Acesso em 11 janeiro 2020.

https://www.sciencedirect.com/topics/computer-science/architecture-description-language. Acesso em: 11 janeiro 2020.

https://blog.prototypr.io/software-documentation-types-and-best-practices-1726ca595c7f. Acesso em 11 janeiro 2020.

https://www.stickyminds.com/article/improve-tester-developer-relationships-helpful-feedback Acesso em 11 janeiro 2020.

https://www.guru99.com/how-to-organize-a-test-team.html. Acesso em 11 janeiro 2020.

https://www.win.tue.nl/~wstomv/edu/2ip30/references/Kruchten4+1.pdf. Acesso em 11 janeiro 2020.

https://softex.br/mpsbr/.Acesso em 11 janeiro 2020.

https://www.it-cisq.org/standards/code-quality-standards/. Acesso em 11 janeiro 2020.