**QUALIDADE DE SOFTWARE**

AULA 1

Profª Maristela Weinfurter

**CONVERSA INICIAL**

**FUNDAMENTOS DE QUALIDADE DE SOFTWARE**

Nesta aula, vamos falar sobre processos de qualidade para o desenvolvimento de softwares e sobre o produto de software com qualidade.

A qualidade de software muitas vezes é tratada como uma disciplina dentro da engenharia de software, no entanto, não há como falar de engenharia de software sem qualidade e vice-versa.

Sendo assim, a tão falada crise de software dentro da área de engenharia de software de certa forma é o gatilho inicial para os primeiros trabalhos, pesquisas e experimentos em relação à aplicação de processos de desenvolvimento de software com critérios de qualidade, considerando a entrega de produtos que atendam às necessidades das partes envolvidas.

Quando falamos em qualidade nesse contexto, não consideramos apenas o código gerado durante o processo, mas também os requisitos, as métricas utilizadas em gerenciamento, entre outras ferramentas e artefatos.

Há um gasto exorbitante quando um produto é feito sem a observação de critérios de qualidade. Considere, assim, que por várias décadas as funcionalidades de códigos têm estado comprometidas, com falta de usabilidade, requisitos não atendidos, além de problemas de implantação e manutenção. Enfim, diferentes momentos do processo vêm comprometendo o produto e o processo. Em especial, o momento de manutenção e adaptação, que gera um retrabalho considerável, acarretando problemas financeiros e outros problemas que nunca se extinguem para engenheiros e desenvolvedores.

Quando um software fica inativo ou funciona precariamente, clientes são prejudicados, são notas fiscais não são emitidas, pagamentos não são feitos, negócios deixam de ser concretizados, aplicativos são substituídos por concorrentes – enfim, o prejuízo é grande.

A forma de melhorar (ou pelo menos minimizar) tais situações é através da implantação de uma boa gerência e de um bom controle de qualidade. O trabalho inicia com os requisitos de um software, e se estende até a sua implantação e manutenção. Para isso, temos hoje normas, padrões, técnicas, ferramentas e muitos profissionais especializados em pensar, planejar e executar atividades relacionadas à qualidade de software.

Grandes empresas já estão na estrada por um bom tempo, com vários aparatos de qualidade em seus processos; porém, quando falamos de empresas pequenas, como podemos estabelecer um processo de qualidade, uma vez que em geral faltam recursos financeiros?

Vamos entender, ao longo da caminhada, que nem tudo está atrelado à gerência de qualidade, mas ainda assim os testes mais simples podem ser feitos pela própria equipe. Tudo é uma questão de aplicar o que é minimamente possível.

Portanto, qualidade de software não tem a ver somente com profissionais de qualidade, pois precisamos considerar todos que trabalham e atuam com desenvolvimento de software.

**TEMA 1 – CONCEITOS DE QUALIDADE DE SOFTWARE**

O termo *qualidade de software* vem sendo lapidado há mais de 30 anos. Ele surgiu em um momento histórico de desenvolvimento de software, justamente para tratar de assuntos inerentes à melhoria, tanto em termos de processos quanto do produto de software. A ideia de qualidade de software ganha maiores proporções nos anos 1990, quando grandes empresas, que faziam uso de vários tipos de sistemas de informações, tais como os ERPs (Enterprise Resource Planning), tiveram que arcar com bilhões de dólares desperdiçados em software, pois eles não entregavam as características e funcionalidades prometidas nos requisitos iniciais (Pressman, 2011).

Como o estudo sobre engenharia de software foi desenhada sobre o desespero de gerentes de projetos de software, analistas de sistemas, engenheiros e programadores, não há como dissociarmos os conceitos de qualidade de software dos fundamentos da disciplina. O debate sobre qualidade de software, por mais abrangente que seja, está intimamente ligado à engenharia de software.

Aliás, vamos voltar um pouco no contexto histórico do software! Aquilo que chamamos de *crise do software* decorre da complexidade inerente ao gerenciamento e ao desenvolvimento de sistemas e aplicações. De forma árdua, considerando o desperdício, com noites sem dormir, finalmente o pessoal de “dev” parou para refletir sobre o que estavam fazendo e como estavam agindo. Mesmo diante de tal episódio, em 1968, levamos mais vinte anos (sim, você leu corretamente, vinte anos!) para perceber que era necessário estabelecer modelos, regras, padrões, critérios e teorias relacionados à qualidade de software.

Finalmente, o nosso pessoal de “dev” olhou ao redor e viu que outras áreas de negócio, especialmente as áreas de engenharia da produção, falavam, disseminavam e utilizavam controle da garantia da qualidade em seus processos fabris. Assim, o engenheiro de software passou a imitar o que havia de melhor nas outras engenharias para a resolução de falhas, erros, projetos não terminados, projetos mal dimensionados e projetos que não tinham um final feliz.

Com esse novo posicionamento na área de desenvolvimento, começaram a aparecer normas e padrões ISO (9000-3, 9126, 15.504, 12.207, 1074, 1298), além de modelos (CMM, CMMI e MPS.BR). Entrelaçada a tais padrões, normas e modelos, a qualidade de software foi passando por um processo evolutivo de detalhamento. Tais detalhamentos fundamentam características importantes, como a preocupação com cultura e ética no processo de engenharia de software, custos de qualidade, melhoria do processo e do produto, segurança de software, verificação, validação, revisão e auditoria, ressignificação do processo de engenharia de requisitos de qualidade de software, caracterização e distinção entre defeitos e erros, gestão da qualidade de software e de processo, além da construção de métricas e ferramentas capazes de dar suporte à qualidade de software. A Figura 1 nos remete a algumas ideias pertinentes aos conceitos fundamentais de qualidade de software.

Figura 1 – Nuvem de palavras sobre itens relacionados ao teste de software



Crédito: master\_art/Shutterstock.

Observem que a nuvem de palavras traz duas em evidência: software e testes. Todas as demais são importantes e fazem parte do nosso ecossistema de qualidade de software, mas, conforme vamos compreendendo melhor sobre o que vem a ser a qualidade de software, percebemos que o assunto “testes” tem conquistado grande importância dentro da ideia de qualidade. Isto fica claro tanto no nível teórico quanto no dia a dia de trabalho dentro das empresas. Porém, não apenas os testes são importantes para a área de qualidade de software, pois há duas referências importantes no mundo da engenharia e da computação que tratam do assunto em questão de forma teórico-experimental: IEEE (Institute of Electrical and Electronics Engineering) e ACM (Association for Computing Machinery). O IEEE é responsável por promoção da engenharia de criação, desenvolvimento, integração, compartilhamento e conhecimento aplicado a tudo que se refere à ciência e a tecnologias de eletricidade e informação. A ACM é uma sociedade para educadores, pesquisadores e profissionais de computação. Hoje é a maior sociedade de computação do mundo, fortalecendo a voz coletiva da profissão, com padrões que ajudam no reconhecimento da excelência técnica.

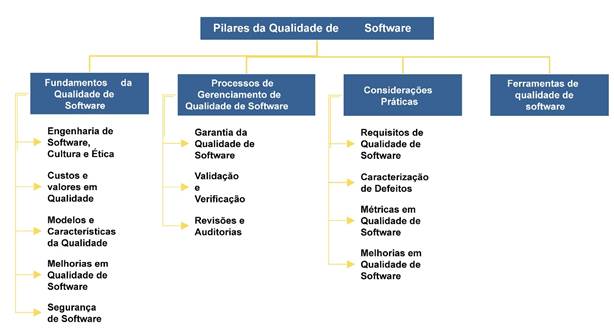
A IEEE e a ACM patrocinaram a criação do guia SWEBOK – “Guide to the Software Engineering Body of Knowledge” (IEE Computer Society, 2022). Esse guia é uma iniciativa para a criação de processos e métodos para a Engenharia de Software, oferecendo uma classificação de suas áreas de conhecimento. Ele divide a Engenharia de Software em dez áreas, entre elas a qualidade. A Figura 2 estabelece algumas subáreas da qualidade de software, segundo o SWEBOK.

Até aqui, falamos de qualidade de software, mas é importante refletir sobre o que é a qualidade. Qualidade, em termos gerais, refere-se às características desejáveis de quaisquer tipos de produtos ou serviços. Tais características precisam ser medidas e aferidas para que possam ser garantidas. Quando falamos de qualidade de software, não é diferente. Ela envolve processos, ferramentas e técnicas que devem ser utilizados para que tais características sejam alcançadas. A conceituação é tão ampla que às vezes acaba sendo divergente. Nesse contexto, há quem diga que a qualidade de software deve estar em conformidade com os requisitos, considerando ainda níveis excelentes de aptidão para uso. Finalmente, é um aspecto que deve ser orientado para o mercado, ambiente em que o cliente é o árbitro.

Resumidamente, a qualidade de software deve atender aos requisitos estabelecidos com precisão, permitindo que as partes interessadas sejam atendidas em suas expectativas e desejos. Para que isso ocorra, é necessário que o processo de desenvolvimento do software esteja baseado em um plano de garantia da qualidade para todas as etapas do projeto, prevenindo e eliminando possíveis erros e defeitos. Lembramos aqui que a qualidade não é apenas uma fase dentro do ciclo de desenvolvimento de software, mas sim um conceito que nasce desde a primeira ideação do software e se estende até a entrega final, com continuidade, considerando ainda a manutenção posterior à entrega do projeto.

A Figura 2 que traz vários pilares que dão sustentação ao mundo da qualidade de software, o que nos ajuda a compreender a amplitude desse contexto.

Figura 2 – Pilares da qualidade de software



O pilar **Fundamentos da Qualidade de Software** aborda questões relacionadas a atividades de planejamento, execução e melhoria contínua dos processos e do software. É subdividido em:

* engenharia de software, cultura e ética;
* custos e valores em qualidade;
* modelos e características da qualidade;
* melhorias em qualidade de software; e
* segurança de software.

Quando pensamos em **Processos de Gerenciamento de Qualidade de Software**, o nosso foco são técnicas e procedimentos que suportam a identificação de erros em artefatos de software. Como principais subtópicos, temos:

* garantia da qualidade de software;
* validação e verificação; e
* revisões e auditorias.

Dentro do tópico **Considerações Práticas**, trabalhamos com técnicas, procedimentos, medições e com o monitoramento da qualidade do processo e do produto. Os subtópicos são:

* requisitos de qualidade de software;
* caracterização de defeitos;
* técnicas de gerenciamento de qualidade de software; e
* métricas em qualidade de software.

Não menos importante, o tópico **Ferramentas de Qualidade de Software** engloba tudo que pode facilitar a aplicação da qualidade, tanto nos processos de desenvolvimento de software quanto no próprio produto (software) em questão.

Estabelecemos o que é a qualidade em seu formato genérico, analisando a qualidade de software e seus principais pilares. Criar software com qualidade não foi, não é e não será, tão cedo, uma tarefa fácil. Afinal, o processo envolve um conjunto de preocupações técnicas e documentais para que, no final, o produto funcione de forma simples, rápida, eficiente, multiplataforma, adaptável, segura, com falhas tendendo a zero, entre outros aspectos necessários quando usamos um software. **Nossa experiência precisa ser a melhor possível!**

**TEMA 2 – PADRÕES DE QUALIDADE DE SOFTWARE**

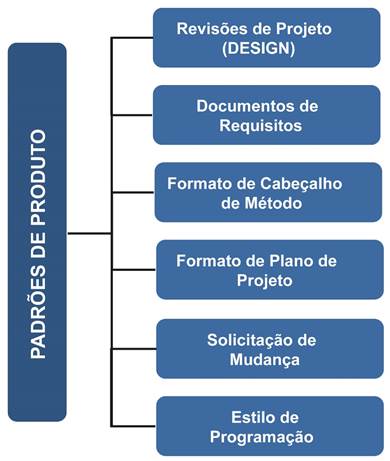
Um dos termos mais utilizados dentro da área de qualidade é o padrão. Padrão, segundo o dicionário, traduz a ideia de que um objeto qualquer pode ser utilizado como modelo para a elaboração de um novo objeto.

Não é diferente na área de software. Padrões de software são importantes porque se baseiam no conhecimento sobre a forma mais adequada e prática para as empresas. Também auxiliam na reutilização da experiência para que os erros do passado sejam evitados. Como a qualidade é algo subjetivo, a utilização de padrões estabelece uma base para a tomada de decisão sobre o nível de qualidade a ser atingido no projeto de software. Além disso, padrões permitem que a continuidade do trabalho por novos engenheiros seja mais tranquila, acarretando menos esforço de aprendizagem para que outros profissionais possam dar prosseguimento às atividades de desenvolvimento de software.

Ao observar essa concepção de padrão, precisamos nos voltar aos padrões estabelecidos dentro da área de qualidade de software, sob dois aspectos.

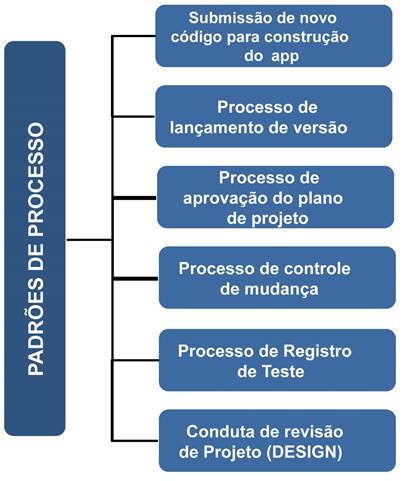
* Padrões de produto (produto = software): estes padrões incluem a documentação, a estruturação dos requisitos, a definição das classes, bem como padrões para a própria codificação dentro de uma linguagem de programação. A figura a seguir trata de alguns exe mplos de padrões de produto.

Figura 3 – Exemplos de padrões de produto



* Padrões de processo (para desenvolver o software): produtos que são desenvolvidos com padrões de qualidade requerem que os processos também tenham garantia de qualidade. Nos padrões de processos, encontramos boas práticas de desenvolvimento, com definições de processos e especificações, validação, e ferramentas de apoio a processos e documentação. A figura a seguir exemplifica padrões relacionados ao processo de desenvolvimento de software.

Figura 4 – Exemplos de padrões de processo



Qualidade de produto e de processo serão inseparáveis em nossa viagem daqui por diante.

Para melhorar a nossa compreensão sobre padrões, vamos partir de um exemplo de aplicação de padrões no desenvolvimento de códigos. Para tanto, vamos utilizar **estilos de programação** dentro de **padrões de produto** (Figura 3). Para exemplificar esse caso, vamos utilizar o PEP 8 – Style Guide for Python. O PEP 8 é um documento que especifica convenções sobre a padronização de códigos escritos em linguagem de programação Python. Ele é muito similar ao PEP7, que foi idealizado para linguagem de programação C. Ou seja, cada linguagem de programação pode ter o seu próprio guia de estilo de codificação.

Voltando ao nosso caso, no início do **PEP 8**, há uma descrição sobre a formatação do código, com três tópicos importantes.

* **Indentação**: nos sugere a utilização do “Python-mode” do Emacs, com quatro espaços por nível de indentação.
* **Tabulações**: sugere que você não misture tabulações e espaços.
* **Comprimento máximo de linhas:** sugere a limitação de 80 caracteres, pois há muitos monitores limitados a 80 colunas. Emacs quebram as linhas em 80 caracteres.
* **Linhas em branco**: funções e definições de classe utilizam duas linhas em branco. Métodos dentro de uma classe são separados com uma linha em branco. Linhas extras para a separação de grupos de funções.
* **Import**: a cláusula **import** deve ser utilizada sempre em linhas separadas.

Vamos utilizar o item 5 da formatação correta de códigos em Python. No Quadro 1, vemos a forma errada e a forma correta. Isto não quer dizer que pela sintaxe a primeira coluna iria retornar algum erro, mas sim que a coluna 2 nos demonstra uma forma mais “limpa” de codificar nessa linguagem de programação.

Quadro 1 – Comparação entre escrita errada e correta na utilização da cláusula import

|  |  |
| --- | --- |
| **Forma errada** | **Forma correta** |
| import sys, os | import sys  import os |

O que vimos até aqui são pequenos exemplos de como a padronização nos encoraja na escrita códigos mais limpos, com a especificação de projetos de forma mais clara e menos ambígua, além do estabelecimento de critérios de reutilização de código, entre tantos outros padrões que nos auxiliam na melhoria contínua do desenvolvimento de software.

**TEMA 3 – MEDIÇÃO DE SOFTWARE**

Suponha que você baixou um aplicativo no seu smartphone, que foi entregue no prazo, dentro do orçamento, executando de forma correta e eficiente todas as funções especificadas. Isso parece definição de felicidade, correto? Nem sempre! Mas como não?

Vejamos:

* O aplicativo pode ser difícil de entender e difícil de ser operado. Isto poderá levar a custos adicionais de manutenção para torná-lo mais simples! Essa situação implica quase uma “nova” construção de várias partes do aplicativo.
* O aplicativo pode ser induzido a um uso indevido, com os resultados capazes de gerar problemas para a empresa que o desenvolveu, como problemas de segurança com os dados.
* O aplicativo pode ter sido integrado a uma determinada plataforma. No momento de adaptá-lo a outra plataforma, às vezes temos quase que reescrever o código para que funcione adequadamente. Imagine que o aplicativo funcione muito bem no Android, porém, por mais que se tenha optado por um framework multiplataforma, foram utilizadas muitas características de baixo nível para acessar recursos do smartphone. Nessa situação, é provável que uma equipe precise modificar e atualizar determinadas funcionalidades para que a mesma aplicação execute no iOS.
* Para encerrar, alguns usuários gostariam de utilizar essa aplicação apenas na web. Sendo assim, estamos falando de outra equipe, que adapta o mesmo aplicativo para que seja executado adequadamente em outros tipos de equipamentos, como notebooks e desktops.

Compreendemos, assim, que nem tudo está associado a custos, prazos e entregas, pois precisamos garantir um gerenciamento adequado, minucioso, ágil e com qualidade total.

Vamos agora à contextualização de métricas necessárias para processar os critérios de qualidade no desenvolvimento de software.

**3.1 CONTEXTUALIZAÇÃO DE MÉTRICAS NA HISTÓRIA**

Quando falamos que o tema da qualidade de software vem de longa data, precisamos voltar mais de 30 anos no tempo, muito antes do desenvolvimento do atual ecossistema tecnológico em que estamos imersos.

Mesmo com livros ou artigos anteriores a 1976, vamos nos basear em um framework para o estabelecimento de critérios de qualidade de software, a partir do artigo “Avaliação quantitativa da qualidade do software”, de Boehm, Brown e Lipow. (1976). Trata-se de um clássico da ciência da computação na plataforma da acm. Descreve um estudo para o estabelecimento de uma estrutura conceitual com alguns resultados iniciais importantes para a análise das características da qualidade de software. A hipótese inicial era de que, quando conseguimos dar atenção às características da qualidade do software, obtemos economias significativas nos custos em todo o ciclo de vida do software.

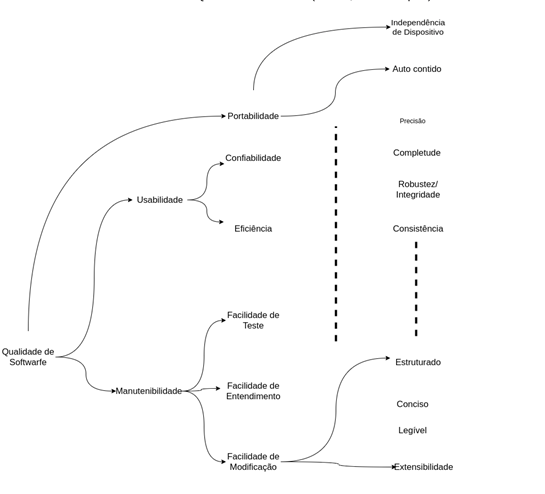
À época, um limitante era a capacidade de avaliação automática e quantitativa da qualidade de software, o que não é mais um problema na atualidade. O artigo traz uma proposta de hierarquia de características, com usos reais para a avaliação da qualidade de software. Com essa hierarquia, foram definidas, classificadas e avaliadas métricas de avaliação de qualidade de software. Um dos benefícios potenciais dessas métricas é a possibilidade de quantificação e a facilidade para a automatização do processo de qualidade.

O estudo definiu, pela primeira vez, uma estrutura clara para questões que por anos foram desafios à qualidade de software. Foi o pontapé para o que veremos adiante. A Figura 5 ilustra esse trabalho brilhante, que deu margem a tantas outras evoluções até o presente momento.

O artigo traz dois pontos importantes e interessantes. Primeiramente, a classificação de 224 erros de software em 13 categorias principais.

* Erros na preparação ou processamento dedados de entrada do cartão.
* Erros de manuseio de fita.
* Erros de manipulação de disco.
* Erros de processamento de saída.
* Erros de processamento de mensagens de erro.
* Erros de interface de software.
* Erros de interface de hardware.
* Erros de interface de base de dados.
* Erros de interface do usuário.
* Erros de computação.
* Erros de indexação e subscrição.
* Erros de procedimentos iterativos.
* Erros de manipulação de bits.

Figura 5 – Árvore das características da qualidade de software



Fonte: Elaborado com base em: Boehm; Brown; Lipow.

Em um segundo momento, foram feitas análises que nos ajudam a compreender os erros de cada fase do desenvolvimento de software. Naquele momento, as fases utilizadas foram:

* definição de requisitos;
* design;
* código e depuração;
* teste de desenvolvimento;
* validação;
* aceitação;
* integração; e
* entrega.

Obviamente, estamos falando de erros remetem aos primeiros sistemas de computadores, que provavelmente executavam em antigos mainframes, ligados à tecnologia daquele momento. No século 21, encontramos muitos pontos em comum, apesar de toda a evolução tecnológica.

Já a análise das fases de desenvolvimento de software não é tão diferente daquilo que fazemos na atualidade, seja em um projeto mais tradicional ou em um modelo ágil.

**TEMA 4 – NORMAS E MÉTRICAS**

As normas internacionais foram estabelecidas com a ideia de responder a uma pergunta fundamental: qual é a melhor maneira de fazer determinada coisa? Por mais que os fatores pareçam óbvios em relação a pesos e medidas, há 50 anos estamos desenvolvendo uma família de padrões que cobrem quase tudo – construção civil, fabricação de sapatos, redes Wi-Fi, desenvolvimento de software, entre tantas outras áreas do conhecimento.

Tais normas têm por objetivo garantir a confiança dos consumidores em produtos que sejam seguros, confiáveis e de boa qualidade. Envolvem padrões para segurança no trânsito, criação de brinquedos e embalagens seguras, que fazendo do nosso mundo um lugar um pouco mais seguro, sendo reguladas por governos e órgãos especiais ao redor do globo.

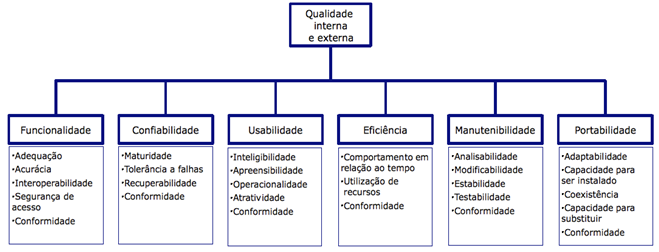
Entre as normas internacionais, temos a ISO/IEC 9126-1:2001, que considera a garantia da qualidade de produto de software. Na versão brasileira, temos a NBR 13596. Elas pressupõem métricas, normas de qualidade, características e subcaracterísticas, além de modelos de processos de avaliação e funções de um software.

A ISO/IEC 9126 define a padronização da avaliação da qualidade de software. O título geral da norma é “Engenharia de software - Qualidade do produto”, sendo alicerçada em 4 pontos importantes:

* modelo de qualidade;
* métricas externas;
* métricas internas; e
* métricas de qualidade em uso.

Basicamente, propõe atributos de qualidade subdivididos em seis macrocaracterísticas, conforme vemos na figura a seguir.

Figura 6 – Características da ISO/IEC 9126-1:2001



Vamos compreender um pouco cada um dos atributos de qualidade, segundo a ISO/IEC 9126-1:2001. Nossa conceituação se baseia em perguntas, o que facilita o nosso entendimento sobre as características e subcaracterísticas.

1. **Funcionalidade**: as funções do software satisfazem as necessidades do usuário? Subcaracterísticas:
   * 1. **adequação**: o software faz o que é adequado?
     2. **acurácia**: o software faz o que é proposto de forma correta?
     3. **interoperabilidade**: o software é integrável a outros sistemas?
     4. **segurança**: o software não autoriza acesso a programas e dados?
     5. **conformidade**: o software obedece a normas e leis?
2. **Confiabilidade**: o software é imune a falhas? Subcaracterísticas:
   * 1. **maturidade**: qual a frequência de falhas por defeitos no software?
     2. **tolerância a falhas**: quando ocorrem falhas, como o software reage?
     3. **recuperabilidade**: é possível recuperar dados em caso de falhas?
3. **Usabilidade**: o software é fácil de ser utilizado? Subcaracterísticas:
   * 1. **compreensibilidade:** o software é de fácil entendimento em relação ao conceito lógico e sua aplicabilidade?
     2. **apreensibilidade**: é fácil de aprender a usar?
     3. **operacionalidade**: é fácil de operar e controlar?
4. **Eficiência**: o software é rápido e simples? Subcaracterísticas:
   * 1. **comportamento durante o tempo**: qual o tempo de resposta e de processamento do software?
     2. **comportamento em relação aos recursos**: o software utiliza muito recurso (em quanto tempo)?
5. **Manutenibilidade**: o software é facilmente modificado? Subcaracterísticas:
   * 1. **analisabilidade**: é fácil de encontrar uma falha quando ela ocorre?
     2. **modificabilidade**: o software é facilmente modificável ou adaptável?
     3. **estabilidade**: há riscos de efeitos inesperados quando é feita alguma alteração?
     4. **testabilidade**: o software é facilmente validado após ser modificado?
6. **Portabilidade**: o software pode ser usado em várias plataformas? Subcaracterísticas:
   * 1. **adaptabilidade**: o software se adapta a ambientes diferentes?
     2. **instabilidade**: é fácil de ser instalado?
     3. **conformidade**: está de acordo com padrões de portabilidade?
     4. **capacidade de substituição**: é fácil de usar em caso de substituição por outro?

O nosso modelo de qualidade segue, em linhas gerais, o que estudamos até aqui. Apresenta várias características que acabam sendo transformadas em métricas, tarefas e metas, que nos ajudam a alcançar a tão almejada qualidade, tanto em nosso produto (software) quanto no processo de desenvolvimento de software.

**4.1 MÉTRICAS, TAREFAS E METAS**

A métrica de software é uma característica intrínseca ao software. Pode ser um elemento de documentação ou um evento, ou ainda uma meta dentro do processo de desenvolvimento de software. Tais métricas podem abordar certas situações, como linhas de código, código compreensível (código limpo), falhas, erros ou quaisquer outras previsões em relação ao software ou ao processo.

As métricas são comumente divididas em três principais tipos:

* **Métrica em relação ao tempo** que um determinado processo leva para ser concluído. Podemos pensar em tempo gasto no processo de desenvolvimento por engenheiros, além de técnicos e calendário de desenvolvimento.
* **Métrica em relação aos recursos** necessários para que um processo seja executado. Neste caso, podemos mensurar o esforço total em número de pessoas por dia, custos de viagens e alocação de recursos em nuvem.
* **Métrica em relação às ocorrências** de um determinado evento, que normalmente incluem erros, defeitos, inspeções de código, número de alterações de requisitos e número médio de defeito por linhas de códigos modificadas.

Como utilizamos essas métricas? De duas formas:

* para atribuir o valor da qualidade do software; e
* para que o padrão de um produto seja identificado.

Anteriormente, estudamos todas as características da ISO/IEC 9126-1:2001. Percebemos que, dentre elas, existem atributos de software, internos e externos. Por exemplo, utilizamos o atributo de qualidade externa de usabilidade. A usabilidade corresponde diretamente a atributos de qualidade internos, como número de mensagens de erro e tamanho do manual do usuário. O atributo externo reusabilidade se relaciona com árvore de herança e tamanho do programa em número de linhas. Com tais atributos de qualidade, poderíamos criar várias métricas para avaliação e inspeções de códigos com base em características externas.

Assim, percebemos que, por mais que certas normas e padrões auxiliem na garantia da qualidade de software, cada empresa estabelece as suas métricas em função de necessidades e urgências na avaliação dos problemas, de acordo com a maturidade do processo de desenvolvimento de software. Não existe uma fórmula mágica para a criação dos indicadores. Eles normalmente são atrelados às necessidades de cada empresa.

Retornando ao produto, conseguimos identificar métricas de previsão para quantificar atributos internos de um software. Tais atributos se relacionam com linhas de código, número de métodos associados a classes, acoplamento e coesão, entre outras características internas.

Segundo Ian Sommerville (2018), as métricas de produtos podem ser subdivididas em métricas estáticas e dinâmicas. As dinâmicas são obtidas durante a execução de um software. As medidas estáticas são pensadas em termos de projeto e documentação do software. O Quadro 2 descreve algumas métricas estáticas de produtos de software.

Quadro 2 – Métricas estáticas de produto de software

|  |  |
| --- | --- |
| **Métrica de Software** | **Descrição** |
| Fan-in / Fan-out | Fan-in: corresponde ao número de métodos que se chama outro método. Quando temos um número alto de fan-in, significa que a outra função está fortemente acoplada ao projeto, de modo que as mudanças terão um efeito dominó.  Fan-out: corresponde ao número de métodos que são chamadas por uma função qualquer. Um fan-out elevado sugere que a complexidade geral da função qualquer é alta, podendo causar complexidade da lógica de controle para chamadas da função. |
| Comprimento de código | Quanto maior o tamanho do código, mais complexo e propenso a erros. |
| Complexidade ciclomática | Complexidade do controle e compreensão do programa. |
| Tamanho dos identificadores | Quanto maiores os nomes dos identificadores, mais significativos são. Com isso mais compreensível, é o programa. |
| Profundidade do aninhamento condicional | Quanto mais aninhamentos de condicionais, mais propenso a erros o código. |
| Índice Fog | Quanto mais palavras uma documentação contém, mais difícil de compreender o código. |

Fonte: Elaborado com base em Sommerville, 2018.

Em geral, as métricas dinâmicas auxiliam na avaliação da eficiência e na confiabilidade do software. Já as métricas estáticas avaliam a complexidade, a compreensibilidade e a manutenibilidade, tanto do software quanto de seus componentes. Além das categorias estáticas e dinâmicas, existem métricas voltadas especificamente para códigos orientados a objetos e componentes, entre outras.

**TEMA 5 – CONFIABILIDADE DE SOFTWARE**

A garantia da confiabilidade de software tem relação com características apresentadas na ISO 9126. Esse modelo de qualidade faz referência aos requisitos do software, agrupando características como funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

Essa discussão está fundamentada na característica de confiabilidade do software. Tais características trazem para si a responsabilidade de garantir ausência de falhas, recuperação de dados após uma possível ocorrência de falhas, emissão de mensagens após uma possível falha, manutenção de padrão de funcionamento após uma possível falha, bem como eficácia no atendimento e o suporte ao software. Tudo isso está relacionado ao desempenho do software e à sua credibilidade com os usuários.

Desde o surgimento da norma ISO 9126, muitos esforços foram aplicados para que a confiabilidade aumentasse com os usuários, trazendo um novo conceito dentro do processo de desenvolvimento de software, o Site Reliability Engineering (SRE). Esse termo foi cunhado pelo Google, em resposta a situações na empresa que colocavam em evidência a confiabilidade do software.

O SRE veio com a missão de proteger, fornecer e melhorar todas as aplicações para que se tornassem escaláveis, confiáveis e eficientes. Junto aos princípios de SRE, surgem elementos para impulsionar a questão da confiabilidade, como maior credibilidade em testes aliados ao CI/CD (Continuous Integration/Continuous Delivery). O CI/CD é um método que permite que a entrega de aplicações seja automatizada, integrada e contínua, ainda com garantia de qualidade.

A área de SRE estabeleceu uma pirâmide com categorias que vão das mais básicas às mais avançadas. O monitoramento é a base da pirâmide, pois estabelece o primeiro momento dos níveis de serviço que se encaixam para metrificar a aplicação. Quais os principais itens a serem monitorados? Latência, tráfego, erros e I/O. O monitoramento pode ser feito via alertas, tickets ou logs.

Uma vez estabelecido o monitoramento, podemos passar para a definição dos comportamentos esperados pela aplicação. Tais comportamentos são medidos pelas métricas SLI (Service Level Indicator), SLO (Service Level Objective) e SLA (Service Level Agreements). O SLI é uma métrica de negócio capaz, por exemplo, estabelecer quantas vendas foram definidas ou quantos acessos um site obteve. A SLO é uma meta colocada no software com base nos indicadores coletas em uma visão técnica, como em falhas diante de *requests*, disponibilidade de API, ou ainda média de tempo de resposta do software. Finalmente, o SLA é um acordo de nível de serviço definido pelos desenvolvedores de software para garantir que a disponibilidade do serviço não incorra em punições e multas judiciais.

Quando falamos em confiabilidade de software, um dos princípios básicos do SRE é um plano sólido para quaisquer problemas que ocorrem em relação a arquitetura, servidor, performance, plano de retorno, monitoramento, segurança, automação e escalabilidade.

A confiabilidade do software está intimamente relacionada ao momento de deploy de uma nova versão ou release do software, sendo geralmente acompanhada por um profissional DevOps, que pode automatizar muitas das atividades com ferramentas como DataDog, Cloudwatch ou Prometheus.

Normalmente, as interrupções de uso de um software se devem a mudanças em sua versão. A SRE tenta melhorar as suas práticas na administração de serviços que garantem troca de versão ou release de forma automatizada, monitorada e segura.

**FINALIZANDO**

Garantir a qualidade de software e de seu processo de desenvolvimento não é uma tarefa fácil. Um bom gerenciamento da qualidade de software preocupa-se em garantir que o número de defeitos seja baixo, de acordo com padrões de manutenção, confiabilidade, portabilidade e tantas outras características mensuráveis.

Percebemos também que os padrões de software nos auxiliam na garantia da qualidade, pois propõem melhores práticas e padrões, que tornam a construção do código mais limpa e eficiente.

Outro detalhe importante é que não é apenas o código que demanda atenção, mas também o gerenciamento de todas as mudanças durante o processo de desenvolvimento. É muito comum entregar um resultado diferente daquilo que foi elicitado nos requisitos, mesmo que com mudanças constantes.

Uma prática que vem crescendo dentro do desenvolvimento de software ágil é o que chamamos de “revisões em pares”, feitas sistematicamente sobre o código. A própria equipe que encarregada pelo desenvolvimento analisa em detalhes possíveis erros e omissões, discutindo e revisando o código.

Às vezes, pensamos em estruturas grandes, com pessoal responsável pela garantia da qualidade, mas na maioria das vezes não é assim. Há um número grande de pequenas empresas, em especial startups, que precisam garantir a qualidade do produto com poucas pessoas. Esses profissionais, além de excelentes programadores, precisam treinar o seu olhar de avaliação e inspeção de código para garantir que o código final esteja de acordo com o nível de satisfação desejado.

Finalmente, se a empresa já tem o ideal, que é uma equipe focada em garantia de qualidade, as medições que coletamos quantitativamente, sobre erros e falhas, podem servir de concepção para métricas que auxiliam na inferência sobre a qualidade de nosso produto e de nosso processo.

O caminho é longo, o caminho é cheio de detalhes, mas podemos começar com pequenos passos. A qualidade começa quando pensamos e refletimos sobre o que é possível adotar para garantir que a experiência do nosso usuário seja incrível.

**QUALIDADE DE SOFTWARE**

AULA 2

Profª Maristela Weinfurter

**CONVERSA INICIAL**

A Garantia e Controle da Qualidade, para a qual utilizamos muito a sigla SQA (Software Quality Assurance), tem dois grandes gatilhos na história:

1. a crise do software e a necessidade de melhoria de processo e produto;
2. a própria garantia e controle de qualidade proveniente das indústrias.

A SQA (Figura 1) é constituída por algumas características, tais como:

* Processos de SQA;
* atividades referentes ao controle de qualidade, e aqui encaixam-se revisões técnicas e outras estratégias de testes;
* utilização de métodos, técnicas e ferramentas de acordo com a engenharia de software;
* controle apurado em cada fase do ciclo de vida do software, bem como de seus artefatos e códigos produzidos e alterados;
* utilização de padrões e normas que se referem ao desenvolvimento e qualidade de software;
* métricas, relatórios e outras formas de visualização do progresso e melhoria dos processos de desenvolvimento de software.

Figura 1 – Software Quality Assurance (SQA)



Créditos: Nadezhda Kozhedub/Adobe Stock.

Para que qualquer processo possa passar por melhorias, é necessária uma boa gestão. Para uma boa gestão, é essencial que tenhamos métricas que mensurem o andamento de cada etapa do processo, bem como seus resultados. Só é possível a gerência de algo sobre fatos e números, e as métricas para a área de qualidade de software é que possibilitam que a garantia da qualidade de software ocorra.

O desenvolvimento de um software já era algo complexo nos anos 1960, porém hoje podemos afirmar que além do aumento de disciplinas envolvidas, também temos uma variedade maior de profissionais que estão trabalhando em conjunto. O desenvolvimento de software tornou-se totalmente multidisciplinar. São analistas de negócios, líderes de projetos, gerentes de produtos, designers de UI, especialistas em experiência do usuário e experiência do consumidor, engenheiros de software, administradores de banco de dados, analistas de testes e todo um conjunto de profissionais relacionados às áreas de negócio para a qual o software se propõe atender.

Como a complexidade só cresce, nossos processos devem ser cada vez mais ajustados à preocupação de garantirmos que o software tenha o mínimo possível de interferência em questões relacionadas à qualidade de software.

**TEMA 1 – VERIFICAÇÃO E VALIDAÇÃO**

Vamos compreender um pouco sobre verificação e validação V&V, um assunto importante dentro do pilar de processos de gerenciamento de qualidade de software. V&V estão intimamente relacionadas com testes de software. Mas qual a diferença entre a verificação e a validação de algo? Parecem até mesmo duas atividades muito similares. Se recorrermos ao dicionário, temos que a verificação está relacionada ao exame da veracidade de algo, a alguma investigação. A validação geralmente segue preceitos preestabelecidos para legitimar-se algo. No entanto, elas de fato são complementares: enquanto a verificação está muito aderente a todo processo de desenvolvimento (requisitos, análise, arquitetura e código), a validação está diretamente relacionada com os testes (unitário, integração, sistema e aceitação).

A verificação estabelece um conjunto de tarefas que irão garantir que um determinado software contenha toda a implementação das funcionalidades de forma correta. Testes unitários, testes de integração ou testes de sistemas fazem parte da verificação. Já a validação propõe um conjunto de tarefas que garante o rastreamento de todos os requisitos desejáveis para o software. Um dos exemplos de testes de validação são os testes de aceitação. A Figura 2 demonstra o modelo V na relação entre a verificação e a validação.

Se fôssemos substituir os conceitos por questionamentos, teríamos algo como:

1. Verificação: desenvolvemos o software da forma correta? Conseguimos atender aos requisitos especificados?
2. Validação: desenvolvemos o software certo?

Figura 2 – V&V: relação entre verificação e validação no contexto de desenvolvimento e testes

Diagrama

Descrição gerada automaticamente

Créditos: Kostiantyn/Adobe Stock.

Vale lembrarmos que dentro da ideia de qualidade de software, não somente o produto (código) é testado, mas também todos os demais artefatos que são produzidos, como documento de requisitos, regras de negócios, diagramas, entre outros.

Com um olhar mais prático sobre o conceito V&V, e talvez com ideias mais práticas do dia a dia, temos alguns exemplos que se encaixam neste contexto de testes:

1. Code Review (estilo verificação): identifica se o desenvolvimento seguiu boas práticas, convenções estabelecidas pela empresa. Este tipo de análise é considerado estático, pois estabelece um olhar sobre o código. O code review geralmente é gerado através de uma equipe com um checklist para apoiar os revisores. Muito importante para o encontro de inconsistências antes das demais etapas de testes.
2. Dev Box Testing (estilo verificação): é uma versão abreviada de Developer Machine. Um engenheiro de testes valida, testa e verifica um recurso na máquina do próprio desenvolvedor. É uma técnica para garantir a qualidade de recursos. Este tipo de teste inicia no ciclo de desenvolvimento, quando as equipes de produtos, Dev e QA ainda discutem tal recurso. Um engenheiro de qualidade verifica os critérios de aceitação (são os pontos de verificação), recursos no escopo e fora do escopo (definição do escopo do teste) e pré-condições, caso haja. Geralmente, é um teste feito entre 10 a 15 minutos, dependendo da complexidade ou dos tipos de defeitos. Aqui, é importante apontar que se a equipe não segue o TDD (Test Driven Development), este tipo de teste pode até ser inútil. Se aplicado corretamente, o esperado é que o produto seja sólido e bem confiável.
3. Pair Programming (estilo verificação): é uma forma de nivelar os colaboradores do time de desenvolvimento, que observa em conjunto tanto regras de negócio quanto boas práticas e padrões de programação. Estilo de verificação intimamente relacionado ao método ágil XP.
4. Testes de Aceitação (estilo validação): são testes nos quais os usuários conseguem criar fluxos não previstos pelo time. A participação das partes interessadas na validação é primordial. O teste de aceitação oportuniza a operação do software num ambiente similar ao de produção.

Voltando ao nosso formato de questionamento, podemos associar algumas perguntas para cada V. Em relação à verificação:

1. Arquitetura: foi avaliada? Em qual nível?
2. Requisitos: são testáveis?
3. Código: segue boas práticas? Padrões? Convenções?
4. Código: foi revisado antes de ser feito commit à branch main?
5. Testes unitários: foram executados com sucesso?
6. Integração contínua: build automático da pipeline quebrou após o commit?
7. Documentação técnica: foi produzida (classes, modelo do banco)?

Já em relação à validação, teríamos:

1. Partes interessadas: concordam com os prazos do projeto?
2. Feedback das partes interessadas: aceitou wireframes?
3. Demonstração: usuários avaliaram a demonstração?
4. Inconsistências: usuários relataram alguma inconsistência nos testes de aceitação?
5. Versionamento: a nova versão teve algum impacto negativo?
6. Documentação: manual é condizente com o software?

Garantir a qualidade de software passa por atividades das mais diversas, como revisões técnicas, auditorias de qualidade, configuração, monitoramento de desempenho, simulações, estudos de viabilidade, revisão de documentação, revisão da base de dados, análise de algoritmos, testes do produto desenvolvido, testes de usabilidade, testes de qualificação, testes de aceitação e testes de instalação.

Vimos que tanto a verificação quanto a validação de software são elementos importantes dentro da garantia da qualidade de software, porém, ainda temos muitos assuntos a serem abordados.

**TEMA 2 – GERENCIAMENTO DA QUALIDADE**

Qualidade de software (Software Quality – SQ) é o grau que um produto de software atende aos requisitos estabelecidos e que depende do grau em que os requisitos representam as necessidades e as expectativas das partes interessadas (Galin, 2018). Enquanto isso, a Garantia de Qualidade de Software (Software Quality Assurance – SQA), segundo o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE), pode ser compreendida como o conjunto de atividades que definem e avaliam a adequação do processo de software para fornecer evidência que estabeleça a confiança de que os processos de software são apropriados e produzem produtos de software de qualidade adequada.

Um atributo chave do SQA é a objetividade da função SQA em relação ao projeto. A função SQA também pode ser organizacionalmente independente do projeto, ou seja, livre de pressões técnicas, gerenciais e financeiras do projeto.

Trabalharemos agora conceitos e elementos importantes para a compreensão do gerenciamento da qualidade de software. Algumas abreviações serão muito utilizadas daqui por diante. Entre elas, encontram-se Software Quality Assurance (SQA), ou melhor, Garantia da Qualidade de Software.

Na sequência, vamos discutir em alguns fundamentos importantes para SQA.

**2.1 FUNDAMENTOS DA SQA**

A SQA se caracteriza por alguns aspectos importantes:

* Planejamento e implementação de uma série de atividades integradas em várias etapas do processo de desenvolvimento de software. Atividades estas que são utilizadas para comprovação da confiança do cliente de que o produto de software atenderá a todos os requisitos técnicos.
* Referência em manterem-se os requisitos técnicos especificados, bem como adequações conforme as necessidades das partes interessadas durante o processo de desenvolvimento. Isso não inclui a qualidade de serviços de operações.
* Adequação técnica do processo de desenvolvimento, mas sem considerar atributos importantes, cronogramas e manutenções de orçamento.
* Cronograma e orçamento do projeto são quesitos importantes na SQA para realização de revisões de contrato, de planejamento de projetos e controle do andamento do projeto.
* Relações entre qualidade do produto, cronograma de projeto e orçamento de projeto, nas quais falhas de cronograma e orçamento ocorrem, são falhas de qualidade de software quase que inevitáveis.

**2.2 ATIVIDADES SQA**

As atividades de SQ referem-se tanto a aspectos funcionais quanto gerenciais no desenvolvimento e manutenção de software. Atividades como:

* Garantia de um nível aceitável de confiança de que o produto (software) e os serviços de operação estarão em conformidade com requisitos técnicos funcionais e sofrerão adequações necessárias.
* Garantia de um nível aceitável de confiança de que o processo de desenvolvimento e operação estarão em conformidade com as necessidades de cronograma e orçamento.
* Gerenciamento de atividades que melhorem e aumentem a eficiência do desenvolvimento de software, operação de software e atividades de SQA. Melhorias em relação à redução de custos são interessantes.

**2.3 PRINCÍPIOS DA SQA**

Segundo a ISO 9000, quando falamos em garantia da qualidade de software (SQA), precisamos compreender como produtos e serviços relacionados a software satisfazem as necessidades das partes interessadas. E tais necessidades envolvem:

* foco no cliente;
* boa gestão;
* envolvimento de técnicos e usuários;
* processos bem claros;
* gestão dos processos;
* melhoria contínua;
* decisões baseadas em fatos e informação;
* relacionamento bom com fornecedores.

Torna-se mais claro, quando olhamos detalhadamente cada um dos elementos que compõe a SQA, segundo Pressman (2011):

* Padrões: IEEE, ISO, ABNT.
* Auditorias e revisões: as revisões técnicas apresentam um relatório de erros. Auditoria, também considerada um tipo de revisão, assegura que as diretrizes sejam seguidas.
* Testes: objetivo principal detectar erros no software e nos demais artefatos do projeto.
* Coleta e análise de erros/defeitos: servem para melhorias em relação aos processos de desenvolvimento e de testes, através da criação de métricas e medições.
* Gerenciamento de mudanças: a todo instante, códigos e requisitos podem mudar. Com isso, o produto está em constante mudança, necessitando um bom controle das versões e releases.
* Educação: melhoria das habilidades e conhecimentos de técnicos envolvidos nos projetos.
* Gerência de fornecedores: muitas vezes, utilizamos APIs, ferramentas, bibliotecas, entre outros pacotes que podem afetar diretamente nossos produtos. É necessária a boa gestão de tais recursos.
* Administração da segurança: proteção de dados e vulnerabilidades dos produtos de software.
* Proteção: todo software de alguma forma envolve o uso por pessoas, e dependendo da sua finalidade, pode gerar consequências mais brandas ou mais catastróficas.
* Administração de riscos: todo projeto envolve riscos, e eles devem ser previstos através de planos de contingência.

**2.2 ERROS, DEFEITOS E FALHAS**

Quando há quaisquer problemas num aplicativo, usamos dizer que há algum “bug” no app. Mas um bug pode ser muito genérico, e é importante entendermos que um software pode conter erros, defeitos ou estar sujeito a falhas.

Melhorando nossa compreensão sobre erros, defeitos e falhas, temos que:

* Erro: ocorre devido a alguma ação humana em consequência de um defeito no software;
* Defeito: ocorre em decorrência a problemas de informações, de dados ou instruções incorretas;
* Falha: o software não se comporta conforme os requisitos estabelecidos ou apresenta resultados inconsistentes.

**2.3 CAUSAS DOS ERROS EM SOFTWARE**

Os erros geram nossos problemas de qualidade de software. Sendo assim, a melhor coisa a fazermos é investigarmos suas causas, com a finalidade de nos prevenirmos.

Podemos classificar os erros de software de acordo com o ciclo de vida do desenvolvimento de software. Falaremos sobre 9 tipos de erros a seguir:

1. Problemas na definição dos requisitos: por mais que tenhamos evoluído na engenharia de requisitos, definições incorretas ainda continuam gerando uma gama grande de erros no software. Às vezes, por definições errôneas, ou por falta de requisitos essenciais ou até mesmo por definições incompletas.
2. Falhas de comunicação: a comunicação é algo imprescindível durante todo processo de desenvolvimento de software, tanto em nível das partes interessadas quanto da equipe técnica. Portanto, instruções não compreendidas no documento de requisitos, ou nas alterações dos requisitos, bem como falta de atenção às mensagens do cliente em relação às mudanças de requisitos, são alguns dos fatores que podem gerar sérios problemas no software.
3. Desvios nos requisitos de software: mesmo que haja compreensão e boa definição nos requisitos, ainda assim, por motivos diversos, a equipe de desenvolvimento pode sair fora do contexto dos requisitos. Por exemplo, a reutilização de módulos ou bibliotecas de projetos anteriores sem análise de aderência ao novo software, omissão de funções por conta de prazos e orçamentos ou melhorias no próprio software sem aprovação gerencial ou do cliente.
4. Erros de projeto lógico: esses erros entram no software durante o projeto do software, através dos profissionais de TI. São definições de requisitos associados a algoritmos errados e definições de processos com erros de sequenciamento.
5. Erros de codificação: é mais fácil do que parece a inclusão de erros durante a codificação. São documentações mal compreendidas, erros linguísticos nas linguagens de programação, erros de aplicação de condicionais e repetições.
6. Não conformidade com documentação: quando o software não está em conformidade com a documentação de projeto, isso dificulta o trabalho das equipes de testes e manutenção, o que compromete a execução do software de forma correta, bem como suas alterações.
7. Falhas no processo de testes: quando o processo de teste é falho, a taxa de erros deixados para trás geram deficiências no software. Isso ocorre quando planos de testes estão incompletos, documentações e relatos de erros e falhas com problemas, a falta de correções imediatamente à detecção de falhas detectadas, bem como correções incompletas de erros detectados por falta de tempo ou pressões.
8. Erros de UI (User Interface): erros na interface do usuário e no procedimento podem causar falhas de processamento mesmo em casos de design e codificação sem erros.
9. Erros de documentação: erros encontrados na documentação das equipes de desenvolvimento, em manuais do software ou em documentos integrados ao corpo do código são algumas situações deste tipo de erro.

O gerenciamento da qualidade envolve o conhecimento minucioso de todas as possibilidades de problemas que usualmente ocorrem num software. Conhecendo os problemas por trás de um software, o gerenciamento da qualidade se torna mais eficiente.

**TEMA 3 – CONTROLE DA QUALIDADE**

O controle de qualidade de software é relacionado a atividades de avaliação da qualidade do produto final de software. O objetivo principal é reter qualquer produto que não se qualifica. Por outro lado, podemos dizer que o principal objetivo da garantia da qualidade de software é minimizar o custo de garantir a qualidade com uma variedade de atividades de infraestrutura e atividades realizadas ao longo dos processos de desenvolvimento de software.

Essas atividades visam prevenir as causas dos erros e detectar e corrigir erros que possam ter ocorrido o mais cedo possível, trazendo a qualidade do produto de software a um nível aceitável. Como resultado, as atividades de qualidade reduzem substancialmente a probabilidade de que os produtos de software não se qualifiquem, na maioria dos casos, além de reduzir os custos de garantia da qualidade.

Podemos dizer que o controle de qualidade aborda um conjunto de ações da engenharia de software para ajudar que um produto de software atinja seus objetivos e metas de qualidade.

Quando ouvimos falar sobre controle de qualidade e garantia de qualidade, parece-nos à primeira vista que significam a mesma coisa. Porém, isso não é verdade. O importante aqui é termos clareza de que a garantia da qualidade tem foco na prevenção dos erros que ocorrem durante o processo de desenvolvimento. Enquanto o controle de qualidade é uma etapa dentro do processo de desenvolvimento de software que permite que o produto de software seja testado de acordo com padrões na busca de erros no software.

Com o controle de qualidade, conseguimos prevenir vários defeitos antes do lançamento do produto. Ele encontra-se no estágio final do processo, e para tanto utilizamos algumas ferramentas e/ou profissionais especializados para revisar a qualidade do software.

As principais técnicas utilizadas no controle de qualidade encontram-se em testar e revisar. Isso envolve a aplicação de:

* Testes unitários (teste estrutural - caixa-branca);
* Testes de integração (teste estrutural - caixa-branca);
* Testes de sistema (teste funcional - caixa-preta);
* Testes de aceitação (teste funcional - caixa-preta);
* Testes não funcionais (usabilidade, carga, segurança, confiabilidade, escalabilidade);
* Testes de regressão (manutenção, confirmação);
* Revisões de requisitos e conceitos;
* Revisões de código;
* Revisões no deployment (Implantação).

**3.1 TESTES UNITÁRIOS**

Os testes unitários fazem parte da fase de testes dentro do ciclo de vida do software e têm por finalidade básica testar individualmente o código. Neste momento, temos por objetivo principal o isolamento de partes do software com o intuito de garantir que cada funcionalidade esteja de acordo com o especificado.

Algo extremamente importante no teste unitário é o planejamento. Ou seja, não são testes feitos ao acaso, mas requerem que se pense nos requisitos de cada funcionalidade a ser testada. Cada planejamento de teste unitário deverá corresponder ao esperado de acordo com o fluxo de dados envolvido naquele processamento.

A responsabilidade dos testes unitários fica a encargo dos programadores, que após codificar uma classe, deveriam aplicar um teste unitário.

Quando temos uma área de qualidade estruturada na empresa, é importante que então um analista de qualidade (QA) crie os conjuntos de testes unitários, que auxiliarão no melhor desempenho do software. Caso contrário, ficará a encargo dos próprios desenvolvedores criarem seus testes.

O princípio de um teste unitário é a comparação dos resultados das funcionalidades a serem testadas com os resultados esperados (planejados).

Vamos verificar algumas ferramentas específicas para algumas linguagens de programação. Obviamente, é somente uma amostra de todos os frameworks para as muitas linguagens de programação existentes.

Para a linguagem:

* JavaScript: temos Jasmine, Jest, Qunit como exemplos. O Jasmine trabalha com o conceito de BDD, no qual o teste é orientado ao comportamento. O Jest foi desenvolvido pelo Facebook para trabalhar com Mocks e testes no React. Já o QUnit é um framework usado pelo jQuery.
* PHP: temos o PHPUnit, que cria testes unitários.
* Python: temos Pytest e Unittest como exemplos. O Pytest é um dos frameworks mais utilizados e possui várias integrações. O Unittest é um framework de testes unitários inspirado no JUnit e funciona de forma semelhante a este, contendo estruturas de teste de unidades existentes de outras linguagens. Suporta orientação a objetos, casos de testes.
* Java: temos o JUnit, que é um dos frameworks mais antigos e que serviu de inspiração para tantos outros existentes no mercado de desenvolvimento. Este fornece uma API que constrói os testes e aplicações gráficas.

Testes unitários então fazem parte dos testes iniciais do código e devem ser os mais atômicos possíveis, fazendo com que cada biblioteca, classe e código sejam testados em detalhes.

**3.2 TESTES DE INTEGRAÇÃO**

Os testes de integração, como o próprio nome já deixa claro, são os testes que vamos fazendo na integração entre todas as funcionalidades já testadas nos testes unitários, considerando agora as requisições HTTP, servidores, SGBDs, APIs externas, gerações de arquivos, envio de mensagens, entre tantas outras ações que possam ser feitas no conjunto de funcionalidades totais do nosso software.

Além de obviamente ser importante no momento do desenvolvimento de um novo software, é também, não menos importante, no momento de quaisquer manutenções ou evoluções no software, pois a probabilidade de alguma nova funcionalidade ser modificada e alterar o comportamento de outras partes já existentes é consideravelmente grande.

Assim como quaisquer testes, os testes de integração podem ser feitos de forma manual, mas também através de ferramentas. Estas, open-source ou não, facilitam e agilizam o processo de testes.

Algumas ferramentas que executam testes de integração: jsdom, Cypress, SuperTest, Postman e Swagger. Observando que, geralmente, tais ferramentas não propiciam somente um ambiente automatizado ou manual para testes de integração, mas também para outras formas de testes e documentação.

**3.3 TESTES DE ACEITAÇÃO**

Testes de aceitação são testes gerenciados e planejados com os mesmos cuidados de um teste de sistema (ou E2E). Normalmente, escolhem-se casos de testes utilizados no teste de sistema para testes de aceitação formais. No entanto, há vários testes de aceitação ditos informais, que não seguem com tanto rigor uma base de testes de aceitação formal. No primeiro caso, devido à formalidade, os critérios de aceitação são conhecidos. Já no segundo caso, eles são feitos de forma subjetiva.

**3.4 TESTES NÃO FUNCIONAIS**

Os testes funcionais descritos na ISO-25010 são ainda pouco explorados pelas áreas de qualidade de software, mas em linhas gerais estes testes não estão associados a funcionalidades, mas em restrições. Verificam os requisitos, tais como escalabilidade, desempenho, usabilidade, confiabilidade, segurança e outras restrições não funcionais que estão relacionadas à execução do software.

**3.5 TESTES END-TO-END (E2E)**

A ideia do teste E2E é a verificação do comportamento do software de ponta a ponta. Basicamente, tenta simular as atividades do usuário final, só que num ambiente preparado similar ao de produção. Geralmente, é a última atividade de teste antes que o produto entre em produção. Na literatura, ele é encontrado como teste de sistema.

A Figura 3 relaciona os conceitos do teste de E2E com sua aplicação vertical e horizontal.

Figura 3 – Pirâmide de testes End-to-End

Diagrama

Descrição gerada automaticamente

Fonte: Elaborado com base em Rocha, 2021.

**3.6 TESTES BETA**

Tipo de teste menos rigoroso, pois os testes são feitos pelo uso de uma versão beta do software e o usuário final é responsável por criar o próprio ambiente, selecionar dados, determinar as funções e recursos que irá testar. Geralmente, o usuário final não tem conhecimento técnico para relatar os problemas encontrados.

**3.7 TESTES DE REGRESSÃO**

Tipo de teste aplicado quando o software sofre alterações que possam levar a gerar bugs. Passa por um roteiro de testes para testes funcionais, desde que não seja tão grande. É aconselhável a utilização de ferramentas que os automatizem. A ferramenta Selenium é um dos exemplos encontrados para fazer este tipo de teste de caixa-preta. Além do Selenium, é possível utilizar ferramentas de cobertura de testes com a finalidade de identificação do percentual do código que foi coberto nos testes. Algumas ferramentas de métricas para testes: OpenClover, IntelliJIDEA.

**3.8 REVISÕES DE REQUISITOS E CONCEITOS**

A finalidade das revisões de requisitos é garantir de maneira formal que os requisitos e seus artefatos estejam em conformidade com a visão das partes interessadas possuem no software. As entradas deste tipo de revisão podem ser casos de negócio, plano de iteração e requisitos do software. A saída será um relatório contendo o registro de revisão.

**3.9 REVISÕES DE CÓDIGO**

As revisões de código objetivam avaliar o código para encontrar erros de quaisquer tipos. Esta tarefa pode ser efetuada por uma equipe ou então por um desenvolvedor mais experiente. O importante neste processo é o feedback dado durante o processo de revisão, sugerindo alternativas, pontos positivos do código, adaptações aos padrões e boas práticas, entre outras questões relativas à escrita do código.

Uma técnica associada à metodologia ágil XP é a atividade em pares. O hábito de programadores trabalharem em pares gera afinidades e complementação de um com o outro em relação a tudo que precisa ser revisado. Existem também ferramentas que podem assistir a organização das tarefas, mediar a conversa entre revisor e desenvolvedor, bem como avaliar a eficácia do processo com métricas. Algumas ferramentas utilizadas são GitHub, Reviewable e Review Board.

**3.10 REVISÕES NO DEPLOYMENT (IMPLANTAÇÃO)**

As revisões no deployment estão associadas ao que chamamos de *CI/CD* (continuous integration/continuous delivery), que é um método para entrega de software com frequência e automatizada. No CI/CD, encontramos os conceitos de *entrega* *e* *implantação contínuas*, garantindo que a implantação do novo software ou das alterações efetuadas ocorram de forma rápida, segura e correta. Estas revisões podem ser feitas manualmente ou automatizadas. O controle de versões dos códigos é gerenciado por ferramentas como o git, por exemplo. O build, os testes e a própria integração podem ser automatizados, evitando com isso que devs e devOps precisem agendar horários para proceder ao trabalho de revisão e deployment. O processo automatizado gera relatórios de quebras do build quando ocorrem falhas. Falhas que podem ser commits esquecidos, versões de bibliotecas de códigos, entre outras situações.

Até aqui, analisamos algumas técnicas e revisões utilizadas dentro do controle de software. Compreendemos que o controle de software possui uma ampla gama de técnicas, ferramentas e métricas que podem ser utilizadas para nos auxiliar na garantia da qualidade do processo de desenvolvimento de software e do produto em si.

**TEMA 4 – MELHORIA CONTÍNUA**

O termo *melhoria contínua* trouxe conceitos da Gestão de Qualidade Total (Total Quality Management – TQM) proposta por Deming. Utilizando-se do ciclo de melhoria contínua chamado *PDCA* (plan, do, check, action) (Figura 4), os processos de desenvolvimento de software adotam então a mesma ideia para melhorar suas atividades.

Várias normas e modelos de maturidade do software, tais como ISO, ABNT, CMMI e MR-MPS tratam questões sobre a melhoria contínua. Tais normas e modelos sugerem que tudo o que for mais crítico dentro dos processos e atividades, devem ser melhorados, refinando-se processos e melhorando-os, gerando maior maturidade dos processos.

Figura 4 – Melhoria Contínua utilizando PDCA

Diagrama

Descrição gerada automaticamente

Créditos: Elnur/Shutterstock.

Quando falamos de uma empresa já consolidada, com muitos anos no mercado, é mais fácil imaginarmos processos, métodos, técnicas e outros recursos sendo utilizados pelas equipes de desenvolvimento e qualidade de software. Porém, a realidade de grande parte das novas empresas, principalmente as que chamamos de *startups*, não é bem assim. Geralmente, tais empresas surgem com uma ideia, dois ou mais sócios e uma equipe de desenvolvimento muito reduzida — por que não dizer, muitas vezes, com um único dev? O objetivo da startup é desenvolver o que chamamos de *MVP* (Minimum Viable Product), ou seja, um pequeno protótipo funcional para que o mercado comece a conhecer sua ideia através de um software que ainda é muito simplificado, que não pôde contar com uma equipe de qualidade de software, com padrões e normas sendo implementadas com consistência. Isso se dá porque é humanamente impossível um único ou poucos desenvolvedores implementarem o software do zero e ainda implementarem todos os processos necessários.

Porém, a ideia de toda startup é crescer, e conforme ela vai recebendo investimentos e vai crescendo, tem condições de melhorar seu time de dev e implementar seu time de garantia da qualidade de software. E é neste ponto que a melhoria contínua é implementada para que a experiência dos usuários, bem como os resultados que a empresa espera, sejam atingidos.

Porém, vamos nos concentrar em auditorias de software. Estas fazem a validação de processos de gestão de projetos, metodologias de qualidade, orçamentos e prazos dos projetos. As auditorias de software podem se manter em mais alto nível (documentações, requisitos), como o software se comporta em relação ao uso de APIs de terceiros, como também podem descer a nível de códigos, bibliotecas e classes.

Estudaremos em detalhes algumas normas e padrões, tais como:

* ISO/IEC-9000;
* ISO/IEC 12207;
* ISO/IEC 15504 (SPICE);
* MR-MPS;
* CMMI-DEV.

Especialmente os dois últimos modelos, MR-MPS e CMMI-DEV, seguem bem a estratégia de garantir que processos de desenvolvimento e qualidade sejam alvo de melhoria contínua. As normas anteriores, por sua vez, também falam da melhoria para eficiência do processo. No entanto, não nos subsidiam com abordagens de implantação da melhoria contínua.

O Controle Estatístico de Processo (CEP) é uma ferramenta bastante utilizada e difundida no setor de indústria, inclusive dando muitas vezes nome a áreas específicas de acompanhamento e controle do processo fabril. Tal ideia também é utilizada dentro do processo de desenvolvimento de software, por algumas empresas já mais maduras, gerando um conjunto de testes de controle estatístico, com a finalidade de apresentar de forma gráfica e mais rápida, como o desenvolvimento de software tem se comportado num período de tempo.

Há também a utilização de Seis Sigma (SS) como forma pela busca da melhoria contínua. Este recurso, também muito utilizado no meio fabril, foi incorporado ao processo de desenvolvimento de software de algumas empresas. O SS adota a estratégia de identificação, classificação e priorização das ações de melhoria e inovação para que a empresa atinja seu grau de maturidade no processo.

Podemos observar que além de normas e padrões específicos para área de desenvolvimento de software, é possível adotarmos técnicas como CEP, SS, Análise de Causa e Efeito, dentre outras, para o amadurecimento do processo de desenvolvimento.

O importante não é exatamente quais técnicas, normas ou modelos serão adotados, mas quais serão passíveis de serem utilizados de acordo com o tamanho do time. Inevitavelmente, adotar processos de melhoria contínua exigirá mais pessoas para auxiliarem no trabalho.

O sucesso da implantação da melhoria contínua encontra-se muito em perceber o momento de atingir novo nível de maturidade, treinar o pessoal envolvido, obter um número de colaboradores nos times suficientes para que nenhuma etapa seja pulada. Lembrando que para gerenciarmos e controlarmos algo é sempre importante termos números. Sem números, não conseguimos mensurar, sem mensurar, não conseguimos gerir.

**TEMA 5 – AUDITORIA DE SISTEMAS**

A auditoria de sistemas é um processo capaz de coletar e avaliar evidências sobre um determinado software. Procede-se à coleta de informações sobre as regras de negócio, a integridade dos dados e todos os recursos envolvidos, para identificação de irregularidades que podem estar estabelecidas em leis, normas ou padrões, tanto a nível interno quanto externo de empresas.

As auditorias podem ser feitas por equipes internas da empresa, bem como por empresas contratadas para tal atividade.

Os resultados de uma auditoria são entregues através de relatórios, com assuntos relacionados às tecnologias, à eficiência e à eficácia do software, se está em conformidade com leis, normas e padrões, bem como se está sendo operado de forma correta.

Uma auditoria pode ter razões das mais diversas, mas como resultados, após a conferência e operacionalização dos relatórios de auditoria, podemos conseguir que nosso software gere mais confiabilidade, maior integridade, maior segurança, melhor desempenho, entre outras características.

Geralmente, as auditorias estão muito aderentes a processos de governança de TI, e para tanto, algumas normas e padrões podem orientar o trabalho dos auditores. Dentre este conjunto de normas e padrões, podemos citar ITIL, ISO, CobIT, entre outros.

Existem alguns tipos de auditoria que estão além da auditoria de desenvolvimento de software, mas que afetam diretamente muitas aplicações, tais como auditoria de integridade de dados, auditoria de segurança da informação, auditoria de segurança física e auditoria de infraestrutura e operações.

Convergindo ao software em si, temos que a auditoria deve registrar as alterações feitas nos dados por um software, sobre os dados originais antes das transformações pelo software até o momento do resultado final com o dado atualizado dentro do software. Ou seja, a auditoria prevê o fluxo básico de conferir e auditar entradas, processamento e saídas. Seguir todo o fluxo de dados e processos, incluindo operações sobre os SGBDs utilizados. Inclusive, há possibilidade de que algumas transformações sobre os dados, possam ser escritas como triggers e stored procedures dentro dos próprios gerenciadores de bancos de dados. Sendo assim, estas devem fazer parte de todo o registro e auditoria em questão.

Além das entradas, saídas e processamento, ainda é importante que a auditoria avalie e registre situações sobre toda documentação ou demais artefatos que acompanham o software. Por exemplo, se o software possuir um dicionário de dados, um glossário empresarial, ou até mesmo um catálogo de dados, os mesmos deverão ser confrontados com aquilo que está em funcionamento dentro do software.

Assim como a auditoria de quaisquer outros processos dentro de uma organização, a auditoria de software pode se utilizar de um checklist, especificando itens importantes como:

* Quais itens serão auditados dentro do software e dentro do seu ecossistema (artefatos, documentos, SGBDs)?
* Qual a ordem de auditoria sobre os elementos anteriores?
* Quais as perguntas que queremos responder?
* Quais avaliações teremos como opções de respostas?
* Como registraremos todo o processo para conseguirmos os resultados da auditoria?

Mesmo com um checklist em mãos, o processo de auditoria é exploratório. Para tanto, é importante que os envolvidos no desenvolvimento do software sejam envolvidos para melhor compreensão dos profissionais que estão realizando a auditoria.

Um dos propósitos da auditoria de software pode fazer parte do processo de melhoria contínua e, dessa forma, a troca de experiência entre gestores e colaboradores gera um maior aproveitamento nos resultados finais da auditoria.

**FINALIZANDO**

Compreender a diferença entre verificação e validação, mergulhar no tema sobre gerenciamento da qualidade, controle da qualidade, auditoria e melhoria contínua reforça pontos muito importantes dentro da gestão de garantia e controle da qualidade.

Percebemos que os temas acima estão muito relacionados a coletarmos, mensurarmos e analisarmos como o processo e como o produto estão no comparativo com nossos próprios processos e na percepção de nosso cliente final.

Verificamos que nem sempre é possível adotarmos todas as possibilidades de testes e controles em nossas empresas, pois todo o processo de garantia de qualidade também requer a contratação ou alocação de profissionais que ficarão exclusivamente concentrados em tal gerenciamento. Obviamente, teremos toda a colaboração do pessoal de desenvolvimento, que ao introduzir seus testes e outras técnicas, geralmente associadas às metodologias ágeis, conseguirão a diminuição de erros, falhas e outros problemas que só pegaríamos no momento da avaliação pelas equipes de qualidade.

O volume de técnicas, ferramentas e técnicas pode até assustar um pouco, mas o importante é que elas sejam adaptadas conforme o amadurecimento dos processos de desenvolvimento de software. Voltando-nos ao que diz a CMMI, o importante é que possamos evoluir e atingir a maturidade desejada dentro dos processos de controle de qualidade.

Auditorias e melhoria contínua são duas estratégias que corroboram para que todos os demais procedimentos e atividades inerentes à qualidade atinjam sucesso.

**QUALIDADE DE SOFTWARE**

AULA 3

Profª Maristela Weinfurter

**CONVERSA INICIAL**

A garantia da qualidade de *software*, assim como de qualquer outra área do conhecimento, inevitavelmente percorre caminhos de gestão com concentração na área de testes, inspeções, revisões, validações e verificações. Os modelos de qualidade nos auxiliam nessa trajetória de melhoria contínua de nossos processos e produtos. Os modelos a serem abordados nesta aula, *plan*/*do*/*check*/*act* (planejar/fazer/verificar/agir) – PDCA e cadeia de Deming, ISO 12207 (ABNT, 2021), Capability Maturity Model Integration – CMMI e Melhoria de Processos do Software Brasileiro – MPS.br, Personal Software Process – PSP e Team Software Process – TSP e o modelo Software Process Improvement and Capability Determination – Spice, da ISO/IEC 15504 (ABNT, 2008a, 2008b) são métodos, padrões e técnicas que nos ajudam em casos dos mais diversos, na conquista da melhoria contínua e da gestão da qualidade.

Em tempos em que falamos tanto de metodologias ágeis, parece-nos difícil a associação entre atividades que nos parecem tão burocráticas a moldes de gerenciamento de projetos que exigem rapidez e eficiência nas suas entregas. Mas a ideia é justamente amadurecermos todos os processos de desenvolvimento e garantia da qualidade, de tal forma que todo o contexto introduzido ao processo seja realmente rápido, eficiente e atinja resultados de qualidade. Com tais modelos, devemos obter melhorias nos processos, adaptações ou adequações de profissionais envolvidos nos processos, de métodos de trabalho, redução de desperdícios, capacitação e engajamento de profissionais, bem como, no final de tudo, conseguirmos aumentar a produtividade com crescimento da qualidade de processos e produtos.

**TEMA 1 – PDCA E A CADEIA DE DEMING**

Vamos conversar um pouco sobre PDCA e a cadeia de Deming. Como já falamos em outro momento, de fato nada foi criado e imaginado de forma inédita, na área de qualidade de *software*. Tudo aquilo que era bom e funcionava, de uma forma ou de outra, foi experimentado e levado para a engenharia de *software* e qualidade.

Nessa linha de pensamento, temos então o ciclo PDCA. O ciclo PDCA é baseado num modelo desenvolvido por Francisco Bacon, em 1620, porém de fato concretizado como conceito pelo norte-americano Walter Shewhart, nos idos de 1920. Mas sua popularidade ocorreu com William Deming em meados de 1940, que o levou para o Japão, onde o modelo foi aplicado com sucesso na indústria. Pela sua popularização com Deming, esse ficou conhecido como o pai do controle de qualidade, com sua ferramenta de PDCA. Assim como a ideia da ferramenta é contribuir para a melhoria contínua de processos e produtos, a própria ferramenta sofreu várias alterações devido a melhorias na sua composição (Figura 1).

Figura 1 – Cadeia de Deming

Diagrama

Descrição gerada automaticamente

Crédito: Heavypong/Adobe Stock.

Cadeia de Deming, ou simplesmente PDCA, é o caminho para que as metas e objetivos sejam atingidos. Em formato de ciclo, essa ferramenta tem o objetivo de auxiliar na solução de problemas, promovendo controle e melhoria contínua dos processos e produtos ou serviços. A Figura 2 detalha a ferramenta. A criação de metas parece simples, mas demanda objetividade. A ferramenta PDCA auxilia na compreensão de oportunidades de solução para um determinado problema, deixando claro que o foco principal do ciclo é buscar a causa do problema e não suas consequências.

O ciclo proposto pelo PDCA constitui uma forma contínua de controle de todas as fases de um processo de desenvolvimento de *software*. Um fato importante a ser destacado é que é necessário que o ciclo seja realmente feito para promover a melhoria contínua. As etapas do processo são acompanhadas pela ferramenta, a qual garante um diagnóstico completo de todas as etapas do ciclo de vida do projeto.

Figura 2 – PDCA

Diagrama

Descrição gerada automaticamente

Crédito: Dmitry/Adobe Stock.

No primeiro item do PDCA, o planejamento define o que chamamos de *indicadores de desempenho* (KPIs). Para construção dos KPIs existem algumas técnicas e ferramentas que podem direcionar melhor quais KPIs precisaremos elaborar e utilizar em nosso PDCA. Dentre tais técnicas e ferramentas, podemos citar, por exemplo: *balanced scorecard* (BSC), pirâmide da *performance*, *tableau* de Bord, entre outras. A adoção de uma ferramenta ou técnica para auxiliar na construção dos KPIs depende em muito do quão complexos podem ser tais indicadores a serem elaborados. Mas o que importa é que os indicadores demonstrem a melhora dos resultados e se os objetivos estão sendo alcançados.

Uma vez planejado o projeto, passamos para a execução das atividades. Algo que é extremamente comum é que se façam desvios de percurso nessa execução, ou seja, quando não se executa exatamente o que foi planejado. Tal episódio gerará falha em nosso PDCA.

A fase de verificação (*check*) é iniciada juntamente com o planejamento. Cada etapa do PDCA deve vir acompanhada sempre da identificação de possíveis erros, que devem ser solucionados para que as melhorias ocorram e de forma correta.

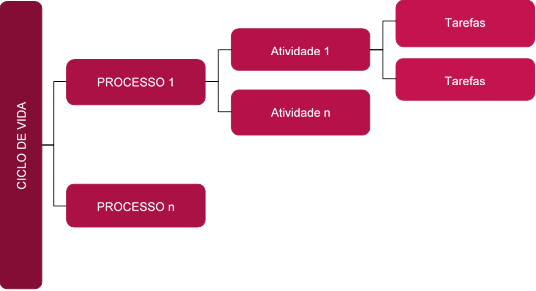
Finalmente, não menos importante, mas por conta do fluxo de atividades anteriores, a etapa de agir (*act*) recomenda que as falhas identificadas sejam corrigidas para que o ciclo feche sua primeira volta. E assim ocorrerá todas as vezes em que o ciclo for reiniciado.

O PDCA é uma ferramenta simples, porém exige que todos estejam empenhados em fazê-la funcionar da melhor forma possível. O que podemos atingir como metas na utilização do PDCA no desenvolvimento de *software* é: motivação e engajamento das equipes, maior responsabilidade sobre rotinas e resultados, melhoria na gestão dos projetos, observação e criação de padrões produtivos, redução de custos, facilidade na identificação de erros e falhas e clarificação para definição de soluções para os problemas encontrados.

**TEMA 2 – ISO 12207**

A ISO 12207 é uma das normas que certifica sistemas de gestão de qualidade. Ela especifica fatores relacionados aos requisitos, nas atividades de desenvolvimento de *software*. O ponto inicial dessa norma foi em 1995, tendo sofrido mudanças em 1998, 2002 e 2004. Em muitos casos, é utilizada em conjunto com a ISO/IEC 15504. Segundo a Associação Brasileira de Normas Técnicas – ABNT, ela é uma norma de fácil entendimento e utilização, enquanto que a ISO 9000 e a ISO 12207 estabelecem padrões mais criteriosos e detalhistas para avaliação e mensuração dos processos. Sendo assim, a ISO 12207 confere maior independência ao uso dos processos e mudanças. Tal norma é muito aderente às fábricas de *software*, que buscam minimizar etapas de gerenciamento do ciclo de vida do *software*. Sua composição é definida por, como demonstrado na Figura 3, processos, atividades e tarefas (ABNT, 2008a, 2008b, 2021).

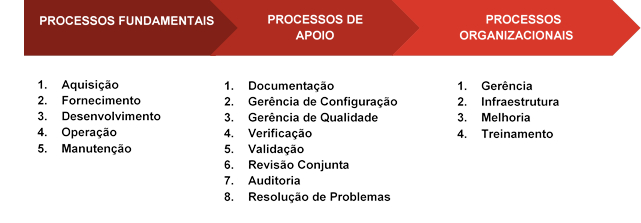
Figura 3 – Estrutura do ciclo de vida na ISO/IEC 12207



Fonte: Elaborado com base em ABNT, 2021.

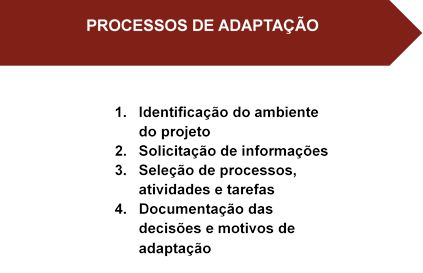
A estrutura do ciclo de vida na ISO/IEC 12207 estabelece que ela é detalhada em processos. A cada processo são atribuídas *n* atividades, que, por sua vez, são especificadas em tarefas. Os processos desse ciclo de vida são padronizados pela sua ordem de execução e utilização e categorizados em processos fundamentais, de apoio e processos organizacionais (Figuras 4-5).

Figura 4 – Processos do ciclo de vida da ISO 12207 (parte 1)



Fonte: Elaborado com base em ABNT, 2021.

Figura 5 – Processos do ciclo de vida da ISO 12207 (parte 2)



Fonte: Elaborado com base em ABNT, 2021.

Vamos detalhar um pouco as funções de cada tipo de subprocesso dos quatro principais processos do ciclo de vida da ISO 12207 (ABNT, 2021).

1. **Processos fundamentais**: são os processos básicos e responsáveis pelo desenvolvimento do *software*. São subdivididos em:
   * **processo de aquisição**: refere-se a todas as aquisições necessárias para atendimento aos clientes, desde compras de prateleiras até ferramentas computacionais;
   * **processo de fornecimento**: refere-se à proposta de revisão e finalização de contratos com fornecedores;
   * **processo de desenvolvimento**: refere-se a tudo que diz respeito ao código, atividades de levantamento de requisitos, análise, testes, implantação e aceitação;
   * **processo de operação**: refere-se às atividades realizadas entre a utilização do *software* e o suporte ao usuário;
   * **processo de manutenção**: refere-se às atividades de solução de problemas do produto.
2. **Processos de apoio**: são os processos ligados aos processos fundamentais e que podem ser aplicados antes ou no meio da utilização daqueles. São subdivididos em:
   * **documentação**: refere-se a toda a documentação produzida para o ciclo de vida do *software*;
   * **gerência de configuração**: refere-se ao gerenciamento de versões dos artefatos produzidos;
   * **gerência da qualidade**: refere-se às verificações que satisfazem os requisitos e a execução dos processos planejados;
   * **processo de verificação**: refere-se às verificações das funcionalidades de cada artefato do projeto;
   * **processo de validação**: refere-se às validações do produto final por meio de vários testes, por exemplo testes unitários e testes de estresse;
   * **processo de revisão conjunta**: refere-se à avaliação em conjunto das verificações do processo e dos artefatos produzidos; também são feitas revisões do gerenciamento de projeto, para que não ocorram desvios de percurso do seu bom andamento;
   * **processo de auditoria**: refere-se a assegurar que as atividades e tarefas sejam realizadas da maneira correta, conforme a norma estabelece;
   * **processo de resolução de problemas**: refere-se aos processos de resolução de conflitos e problemas que possam causar riscos e prejuízos ao andamento do projeto.
3. **Processos organizacionais**: são os processos com atividades relacionadas ao gerenciamento e capacitação de pessoas. São subdivididos em:
   * **processo de gerência**: refere-se à implantação das atividades de gestão de processos e produtos, nos projetos;
   * **processo de infraestrutura**: refere-se à designação da infraestrutura que permita a integração de ferramentas, técnicas, padrões, *hardwares* e *softwares*;
   * **processo de melhoria**: refere-se à avaliação, mensuração, controle e melhoria do processo do ciclo de vida do *software*;
   * **processo de treinamento**: refere-se à capacitação dos profissionais alocados para executarem as atividades de forma rápida e eficiente.
4. **Processos de adaptação**: são os processos que observam a realidade de cada organização e definem seus ciclos de desenvolvimento de acordo com as peculiaridades mais convenientes aos processos do ciclo de vida. São subdivididos em:
   * **identificação do ambiente do projeto**: refere-se à análise e adaptação de fatores relacionados a pessoas, requisitos, *softwares*, políticas de qualidade, pontos estratégicos e cultura organizacional;
   * **solicitação de informações**: refere-se às informações decorrentes do processo de implantação e adaptação dos requisitos da norma ao ciclo de vida do desenvolvimento;
   * **seleção de processos, atividades e tarefas**: refere-se aos processos, suas atividades e tarefas, que devem possuir documentos contendo características e designação de responsáveis por desenvolvê-las e executá-las;
   * **documentação das decisões e motivos de adaptação**: refere-se ao plano de viabilidade adaptativa, com análises de custos e riscos.

A norma ISO/IEC 12207 busca a conscientização e organização das melhorias na gestão dos processos. Como já comentamos, ela é aberta e pode ser adaptada a cada realidade (ABNT, 2021).

**TEMA 3 – CMMI E MPS.BR**

O CMMI é um modelo que contém práticas importantes para a maturidade de disciplinas específicas, tais como engenharia de *software*, engenharia de sistemas, desenvolvimento integrado de processo e produto e escolha de fornecedores. Esse modelo é administrado pelo Instituto CMMI. Ele baseia-se em melhores práticas para o desenvolvimento e manutenção de *software* e está dividido em cinco níveis de maturidade, os quais atestam o grau de maturidade dos processos de uma empresa (Figura 6). Seu foco é melhorar continuamente os processos organizacionais. O CMMI surgiu na década de 1980, tendo sido o ano de 1991 o marco para o desenvolvimento dos Capability Maturity Models –  CMMs, os quais antecedem os CMMIs. A constituição do CMMI tem por base SEI Software CMM (SW-CMM), Electronic Industries Alliances’s Systems Engineer Capability Model (EIA-Secm) e Integrated Product Development CMM (IPD-CMM).

Figura 6 – CMMI

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Crédito: Dizain/Adobe Stock.

O CMMI estabelece vários níveis de maturidade a serem seguidos para que a empresa atinja seu objetivo maior, que é o nível 5 (Figura 7).

1. **Nível 0: incompleto** – os processos não funcionam ou não atingem todas as metas e objetivos definidos pelo CMMI.
2. **Nível 1: executado**– os processos definidos pelo CMMI já estão sendo executados com tarefas que produzem artefatos definidos.
3. **Nível 2: controlado**– todos os critérios estabelecidos no nível 1 já foram satisfeitos e todos os processos estão de acordo com a política definida para a organização. Os colaboradores estão executando seu trabalho com acesso a recursos adequados e as partes interessadas estão envolvidas ativamente nos processos de desenvolvimento. Além disso, as tarefas e o *software* são monitorados, controlados e revisados em conformidade com os processos do CMMI para esse nível.
4. **Nível 3: definido**– todos os critérios estabelecidos no nível 2 já foram satisfeitos e o processo de desenvolvimento é adaptado com base no conjunto de processos, agora padronizados de acordo com a cultura organizacional da empresa. O *software*, a mensuração e outras questões de melhoria do processo já agregam valor ao processo organizacional.
5. **Nível 4: gerenciado quantitativamente**–todos os critérios estabelecidos no nível 3 já foram satisfeitos e o processo é gerenciável, fazendo uso de medição e avaliação quantitativa. Objetivos quantitativos foram estabelecidos e o desempenho do processo é amparado em critérios de controle do processo.
6. **Nível 5: otimizado**– todos os critérios do nível 4 foram satisfeitos e os processos agora são adaptados e otimizados, fazendo uso de meios quantitativos (estatísticos) para atender à mudança de necessidades do cliente e à melhoria contínua do processo.

Figura 7 – Níveis de maturidade do CMMI



A implantação do CMMI leva uma empresa a atingir os níveis desejados em termos de qualidade dos processos e do *software*. Certamente, quando uma empresa decide adotar qualquer padrão de qualidade, há muito recurso financeiro e pessoal envolvidos. Ou seja, é algo que gera investimentos, mas que gera retorno a médio e longo prazo, tanto na qualidade dos processos quanto do *software*.

**TEMA 4 – PSP E TSP**

O **PSP**foi desenvolvido por Watts Humphrey nos anos de 1995, para orientação do planejamento e desenvolvimento de *software*. É um processo aderente à engenharia de *software*, em especial para desenvolvimento de projetos individuais e de pequeno porte. Pode ser compreendido como um subconjunto do CMMI, porém possui uma revisão contínua em cada ciclo do desenvolvimento.

Com a adoção do PSP, conseguimos atingir alguns objetivos, tais como:

1. melhoramos o planejamento e a execução de nossos cronogramas;
2. melhoramos a estimativa de prazos e custos; e
3. estabelecemos o engajamento dos desenvolvedores na melhoria contínua.

A Figura 8 ilustra as etapas do PSP.

Figura 8 – Etapas do PSP

Diagrama

Descrição gerada automaticamente

Crédito: Kostiantyn/Adobe Stock.

O PSP possui níveis conforme as etapas do processo de desenvolvimento de *software*. Tudo se inicia no ***baseline personal process***, detalhado pelas atividades **PSP0** e **PSP0.1**, que correspondem ao nível que introduz o processo corrente, os padrões de código, bem como a metrificação dos processos.

No PSP0 há três procedimentos importantes:

1. planejamento;
2. desenvolvimento; e
3. testes.

No PSP0, os desenvolvedores deverão garantir que a elicitação do projeto foi devidamente coletada e analisada. Conforme o processo avança, vamos para o PSP0.1, que adiciona padrões e boas práticas para codificação, métricas e melhoria do projeto. As atividades **PSP1 e PSP1.1 (*personal planning process*)** correspondem ao nível que recebe os dados coletados no PSP0 e no PSP0.1 e estimam o tamanho do projeto, bem como preparam o relatório de testes para futuras verificações e validações. Com essa estimativa, é possível planejar questões relacionadas ao cronograma necessário para implantação e treinamento, bem como os custos que serão envolvidos no desenvolvimento do projeto. No futuro, o gerente do projeto pode reutilizar essas estimativas para melhorar seu processo de planejamento.

Passadas as atividades 0 e 1, vamos para a ***personal quality management* (PSP2 e PSP2.1)**, fase em que se faz, propriamente, a análise do projeto e a revisão do código. Nessa etapa, é trabalhada de forma preventiva a remoção de defeitos, para que possamos aprender a avaliar e melhorar os processos medidos, bem como são determinadas as tarefas para descoberta e remoção de defeitos do produto. Uma ferramenta útil nesse momento é a construção de *checklists* de verificação, para que as revisões do projeto e do código sejam alinhadas com os objetivos do projeto.

Finalmente, chegamos ao nível **PSP3 (*cyclic personal process*)**, o qual direciona para o uso do **TSP**. O TSP é um processo intermediário entre o CMMI e o PSP, que fornece um arcabouço de processos operacionais que é projetado para auxiliar os times de desenvolvedores, bem como para organização do projeto com a finalidade de produção de *software*. Esses projetos podem variar de pequenos a projetos com milhares de linhas de código (Kloc) e até projetos com milhões de linhas de código.

O foco do TSP é a melhoria dos níveis de qualidade e produtividade do projeto de desenvolvimento de *software* e também do time de desenvolvimento. Compreender melhor os custos e conseguir cobrir todas as atividades de desenvolvimento e qualidade são alguns de seus objetivos. Para que se utilizem os conceitos do TSP, é importante que engenheiros e desenvolvedores conheçam o PSP e seu funcionamento. O líder do time deve conhecer bem todas as etapas envolvidas para que consiga gerenciar tanto o projeto quanto o time. O início do ciclo do TSP se inicia no lançamento, que é conduzido pelo líder treinado e certificado.

Esse lançamento deve ter sido iniciado juntamente com o planejamento do projeto e a formação da equipe, em um momento em que são estabelecidas metas, funções da equipe, avaliados os riscos, estimado o esforço, bem como alocadas as tarefas e elaborado o plano da equipe de desenvolvedores. O plano deve ser semanalmente revisto, para que o *status* do projeto seja atualizado em ciclos de tempos pequenos. Caso haja necessidade de correções de percurso, é mais fácil de se fazer o replanejamento e caminhar para o sucesso do projeto e não para o seu fracasso. Todo ciclo de desenvolvimento termina com um *post mortem*, que avalia o desempenho, as configurações do planejamento e trabalha sobre todas as lições aprendidas. Isso auxilia na melhoria constante do processo de desenvolvimento.

É muito importante que o líder do projeto atue como um *coach*, apoiando a equipe e os desenvolvedores individualmente. Liderar pessoas é muito mais difícil do que liderar apenas projetos. É praticamente uma arte manter a equipe focada e motivada para que todos atinjam os mesmos objetivos. Vale lembrar que o líder do projeto é responsável pelo desenvolvimento individual de cada um e também pelo resultado final do projeto.

**TEMA 5 – SPICE E ISO/IEC 15504**

A norma ISO/IEC 15504 surgiu em 1991 e trazia consigo o pensamento da normatização de conceitos, a fim de facilitar as definições de características dos processos. Na sequência, em 1993, surgiu um projeto chamado Spice, considerando três aspectos: auxiliar o projeto da norma 15504, incluir testes de campo e coletar dados para consolidação da nova norma. Como outras normas e padrões, o seu objetivo é focado na melhoria de processos. O Spice utiliza-se da análise de resultados como pontos fortes e fracos, bem como do gerenciamento de riscos. A Figura 9 nos mostra a estrutura básica da ISO/IEC 15504 (ABNT, 2008a, 2008b).

Figura 9 – Estrutura da ISO/IEC 15504



A norma estabelece partes independentes e critérios diversificados para avaliação de processos de desenvolvimento de *software*. A ISO teve por ideia inicial a definição de um *framework*para mapear e monitorar processos para que as práticas de melhorias fossem viáveis (ABNT, 2008a, 2008b).

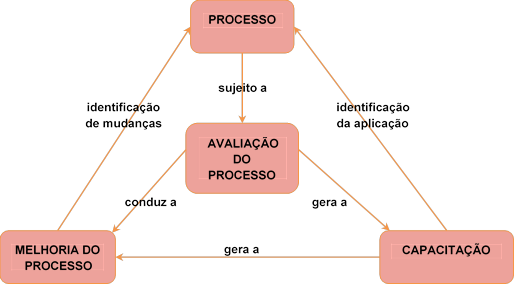
Detalhando agora um pouco mais sobre cada parte da norma, temos, em sua estrutura (ABNT, 2008a, 2008b).

1. **Parte 1 (conceitos e vocabulário)**:traz uma introdução sobre os conceitos e uma avaliação dos processos, com um destaque para o glossário, que define os principais termos avaliativos utilizados.
2. **Parte** **2 (avaliação de processos)**:estabelece quais são os requisitos mínimos, para realização da avaliação, para que haja uma boa granularidade no processo. Nessa fase há a classificação de dados, a designação de responsabilidades e o desenvolvimento de modelos para o acompanhamento dos processos.
3. **Parte 3 (recomendações de avaliação)**:fornece recomendações para avaliação dos requisitos dos processos.
4. **Parte 4 (recomendações para melhoria dos processos)**: dá uma perspectiva para a especificação dos processos, identificando os seus pontos fortes, fracos e os seus riscos.
5. **Parte 5 (exemplo de avaliação)**: essa parte estabelece um exemplo de *process assessment model*(PRM), o qual se resume num modelo de referência de processos de ciclo de vida estabelecido pela norma ISO/IEC 12207 (ABNT, 2021). Além do PRM, há também a exemplificação do *process assessment model* (PAM), que estabelece fatores de avaliação e medição dos processos conforme planejado no cronograma.

O modelo PRM deve ser desenvolvido para explicitar as necessidades das partes interessadas e sua principal característica é descrever os requisitos priorizando as alternativas que permitam a avaliação dos resultados esperados de cada processo. O modelo PAM é semelhante e deve complementar as referências realizadas no PRM, porém ele tem a função mais específica de guiar o monitoramento previsível dos indicadores dos processos. Ele é composto por práticas de base e artefatos produzidos.

Na Figura 10, temos um diagrama da utilização da ISO/IEC 15504.

Figura 10 – Forma de utilização da ISO/IEC 15504



Fonte: Elaborado com base em ABNT, 2008a, 2008b.

O Quadro 1 a seguir identifica as dimensões de processo da ISO/IEC 15504, o que compreende 48 processos organizados em 3 categorias e 10 grupos.

Quadro 1 – Dimensões de processo da ISO/IEC 15504

|  |  |
| --- | --- |
| **Fundamentais** | **Organizacionais** |
| Aquisição  Fornecimento  Engenharia  Operação | Gerência  Melhoria do processo  Recursos de infraestrutura  Reúso |
| **Apoio** | |
| Controle de configuração  Garantia da qualidade | |

Fonte: Elaborado com base em ABNT, 2008a, 2008b.

Cada processo é definido por meio dos seguintes elementos.

1. Identificação: uma forma de identificação do processo.
2. Nome: todo processo deve ter uma denominação.
3. Propósito: finalidade de tal processo segundo o contexto da norma.
4. Resultados: resultados que esperamos durante a implementação do processo.
5. Práticas de base: atividades que, ao serem executadas, contribuem para que o propósito do processo seja atingido.
6. Produtos de trabalho: produtos esperados com a utilização do processo; lista de produtos de trabalho para cada processo, objetivando a melhoria do processo.

Assim como no caso do CMMI, a norma também contempla níveis de capacidades (ABNT, 2008a, 2008b). Sendo assim, temos o que se segue.

1. **Nível 0: incompleto**–nesse caso, o processo não existe ou é falho.
2. **Nível 1: executado** – o processo atinge os seus objetivos, mas sem padrões de qualidade e sem controle de custos e cronograma de atividades.
3. **Nível 2: gerenciado**– nesse caso, o processo é planejado e acompanhado, satisfazendo requisitos definidos para qualidade, prazo e custos. Seus produtos de trabalho são gerenciáveis.
4. **Nível 3: estabelecido**– o processo é executado e gerenciado com melhorias, com base em um processo padrão eficaz e eficiente.
5. **Nível 4: previsível**– o processo é executado dentro dos limites de controle predefinidos, e sua mensuração é detalhada e analisável.
6. **Nível 5: otimizado**– o processo sofre melhoria contínua, de forma disciplinada.

A norma não tem por finalidade definir os passos que devem ser seguidos para elaboração dos modelos PRM e PAM. Ela apenas cita os componentes que devem ser desenvolvidos para o acompanhamento dos processos, destacando-se a avaliação, por meio de referência dos requisitos e medição do cronograma de entrega do produto (ABNT, 2008a, 2008b). Na prática, a avaliação dos processos é conduzida por meio do modelo de avaliação de processos.

**FINALIZANDO**

Analisamos, nesta aula, conteúdos bastante teóricos, mas que auxiliam e proporcionam melhorias incríveis tanto aos processos de desenvolvimento de *software* quanto aos produtos finais. Como falamos anteriormente, a área de qualidade parece-nos, por vezes, enfadonha e burocrática; porém, à medida que avançamos no processo de melhoria e adoção de técnicas, métodos e ferramentas, em um contexto ágil, percebemos o quanto ganhamos por termos processos mais maduros.

Assim como a engenharia de *software*, o aperfeiçoamento pessoal de desenvolvedores e engenheiros de qualidade não é um gasto, mas, sim, um investimento. Garantir que processos e produtos satisfaçam, cada vez mais, todas as partes envolvidas e, especialmente, nosso cliente, remove custos, retrabalhos e muitos problemas altamente evitáveis, durante o processo.

Inicialmente, a adoção de normas e padrões é demorada, pois envolve que todos estejam na mesma “página”, engajados e desejando o objetivo final: qualidade. À medida que as coisas se consolidam, não conseguimos mais imaginar que possamos fazer as atividades sem um contexto padronizado e rico em princípios de qualidade. Coletar, medir e melhorar devem ser os verbos mais utilizados para que nossos produtos encantem, cada vez mais, nossos clientes, para que nossos profissionais invistam mais seu tempo em desafios e menos em retrabalho.

Questão 1/5 - Qualidade de Software

A avaliação da qualidade de software segundo a  ISO/IEC 9126 tem por título geral “Engenharia de software - qualidade do produto”. Esta norma é constituída por 6 subdivisões.

A ISO/IEC 9126 está subdividida em:

Nota: 20.0

|  |  |  |
| --- | --- | --- |
|  | A | Qualidade interna, qualidade externa, processo, engenharia de produto, metodologias e  ferramentas |
|  | B | Funcionalidade, engenharia de produto, metodologias, ferramentas, portabilidade e testes. |
|  | C | Funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.  **Você acertou!**  CORRETA: Conforme texto da Aula 01 no Tema 04 |
|  | D | Qualidade externa, qualidade de processo, qualidade de produto, metodologias, boas  práticas e ferramentas. |
|  | E | Funcionalidade, confiabilidade, processos, engenharia, qualidade e métricas. |

Questão 2/5 - Qualidade de Software

A avaliação da qualidade de software segundo a  ISO/IEC 9126 tem por título geral “Engenharia de software - qualidade do produto”. Esta norma é constituída por 6 subdivisões: Funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

Dentro da Eficiência  encontramos as seguintes características:

Nota: 20.0

|  |  |  |
| --- | --- | --- |
|  | A | Comportamento e qualidade. |
|  | B | Compreensibildiade e qualidade. |
|  | C | Processos e comportamento. |
|  | D | Comportamento durante o tempo e comportamento em relação aos recursos  **Você acertou!**  CORRETA: Conforme texto da Aula 01 no Tema 04  Eficiência: o software é rápido e simples? Subcaracterísticas:  comportamento durante o tempo: qual o tempo de resposta e de processamento do software?  comportamento em relação aos recursos: o software utiliza muito recurso? em quanto tempo? |
|  | E | Qualidade e comportamento. |

Questão 3/5 - Qualidade de Software

A ISO 12207 é uma norma que certifica sistemas de gestão de qualidade. Esta norma especifica fatores relacionados aos requisitos dentro das atividades de desenvolvimento de software. Ela possui quatro níveis principais: processos fundamentais, processos de apoio, processos organizacionais e processos de adaptação.

Quais são os subníveis do nível processos fundamentais?

Nota: 20.0

|  |  |  |
| --- | --- | --- |
|  | A | Identificação do ambiente do projeto, solicitação de informações, seleção de processos,  atividades e tarefas e documentação das decisões e motivos de adaptação. |
|  | B | Documentação, gerência de configuração, gerência de qualidade, processo de verificação,  processo de validação, processo de revisão conjunta, processo de auditoria, processo de  resolução de problemas. |
|  | C | Processo de aquisição, fornecimento, desenvolvimento, operação e manutenção.  **Você acertou!**  CORRETA: Conforme texto da Aula 03 no Tema 2  Vamos detalhar um pouco sobre as funções de cada tipo de Subprocesso dos 4 principais processos  do ciclo de vida da ISO 12207.                Processos Fundamentais são os processos básicos e responsáveis pelo desenvolvimento do  software e subdivididos em:    Processo de aquisição: refere-se a todas as aquisições necessárias para atendimento aos clientes,  desde compras de prateleiras, até ferramentas computacionais.    Processo de fornecimento: refere-se à proposta de revisão e finalização de contrato com  fornecedores.    Processo de desenvolvimento: refere-se a tudo que diz respeito ao código, atividades de  levantamento de requisitos, análise, testes, implantação e aceitação.    Processo de operação: refere-se às atividades entre a utilização do software e o suporte ao usuário.    Processo de manutenção: refere-se às atividades de solução de problemas do produto. |
|  | D | Processo de gerência, de infraestrutura, de melhoria e de treinamento. |
|  | E | Processo de certificação, validação e verificação de erros, defeitos e falhas. |

Questão 4/5 - Qualidade de Software

Dentro dos processos de Verificação e Validação de software temos uma técnica muito interessante chamada Pair Programming, a qual é um estilo de qualidade focada na Verificação. Pair Programming, ou programação em pares, tem por finalidade nivelar os colaboradores de um time de desenvolvedores quanto às regras de negócio, boas práticas e padrões de programação.

Tal técnica está relacionada ao método ágil:

Nota: 20.0

|  |  |  |
| --- | --- | --- |
|  | A | XP (eXtreming Programming).  **Você acertou!**  CORRETA: Conforme texto da Aula 02 no Tema 01  Pair Programming (estilo verificação), é uma forma de nivelar os colaboradores do time de  desenvolvimento que observam em conjunto tanto regras de negócio quanto boas práticas e  padrões de programação. Estilo de verificação intimamente relacionado ao método ágil XP. |
|  | B | SCRUM. |
|  | C | Kanban. |
|  | D | Lean. |
|  | E | FDD. |

Questão 5/5 - Qualidade de Software

Revisões de código avaliam o código para encontrar erros de quaisquer naturezas. É uma tarefa que pode ser desenvolvida em equipe ou apenas por um programador experiente. O que importa mesmo é o feedback dado durante o processo de revisão, sugerindo alternativas, pontos positivos do código, adaptações aos padrões e boas práticas entre outras situações.

A ideia de revisões de código é algo de longa data, porém qual a metodologia ágil que o colocou mais em evidência?

Nota: 20.0

|  |  |  |
| --- | --- | --- |
|  | A | SCRUM. |
|  | B | Kanban. |
|  | C | FDD. |
|  | D | SCRUM e Kanban. |
|  | E | XP (eXtreming Programming)  **Você acertou!**  CORRETA: Conforme texto da Aula 02 no Tema 3.9  As revisões de código objetivam avaliar o código para encontrar erros de quaisquer tipos.  Esta tarefa pode ser efetuada por uma equipe ou então por um desenvolvedor mais experiente.  O importante neste processo é o feedback dado durante o processo de revisão, sugerindo  alternativas, pontos positivos do código, adaptações aos padrões e boas práticas, entre outras  questões relativas à escrita do código. Uma técnica associada à metodologia ágil XP é a atividade  em pares. O hábito de programadores trabalharem em pares gera afinidades e complementação de  um com o outro em relação a tudo que precisa ser revisado. Existem também ferramentas que  podem assistir a organização das tarefas, mediar a conversa entre revisor e desenvolvedor,  bem como avaliar a eficácia do processo com métricas. Algumas ferramentas utilizadas são  GitHub, Reviewable, Review Board. |

Questão 1/5 - Qualidade de Software

O BDD é uma abordagem que funciona bem com métodos ágeis, projetado para que a gestão e a entrega dos projetos de software tenham uma melhor comunicação entre clientes e desenvolvedores. Ele garante que projetos permaneçam sempre focados na entrega do que o negócio realmente precisa, e que todas as necessidades do usuário estejam atendidas. Nesta metodologia, os testes são importantes, mas os testes não são os elementos que conduzem o desenvolvimento. Seu objetivo é que as metas e os resultados para o cliente sejam definidos de forma clara.

Além de todas as características já comentadas, o BDD auxilia em qual atividade de qualidade de software?

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | O método BDD auxilia na construção de um código limpo e utilizando padrões. |
|  | B | O método BDD auxilia na redução de custos no ciclo de vida do software. |
|  | C | O método BDD auxilia na implementação da UI. |
|  | D | O método BDD auxilia na redução de falhas no momento da entrega do software.  CORRETA: Conforme texto da Aula 05 no Tema 2  O BDD, do quadrante 2 segundo Agile Testing, é uma abordagem que funciona bem com métodos  ágeis, projetado para que a gestão e a entrega dos projetos de software tenham uma melhor  comunicação entre clientes e desenvolvedores. Ele garante que projetos permaneçam sempre  focados na entrega do que o negócio realmente precisa, e que todas as necessidades do usuário  estejam atendidas. Nesta metodologia, os testes são importantes, mas os testes não são os elementos  que conduzem o desenvolvimento. Seu objetivo é que as metas e os resultados para o cliente sejam  definidos de forma clara.  …  Todas as User Stories que não possuam critérios de aceitação acabam suscetíveis a falhas durante o  processo de aceite, além de possibilitarem erros do software em relação ao negócio para o qual ele  está sendo construído. O método BDD auxilia na redução de falhas no momento da entrega do  software. |
|  | E | O método BDD é idêntico ao TDD, é orientado a testes. |

Questão 2/5 - Qualidade de Software

Dentro dos processos de Verificação e Validação de software temos uma técnica muito interessante chamada Pair Programming, a qual é um estilo de qualidade focada na Verificação. Pair Programming, ou programação em pares, tem por finalidade nivelar os colaboradores de um time de desenvolvedores quanto às regras de negócio, boas práticas e padrões de programação.

Tal técnica está relacionada ao método ágil:

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | XP (eXtreming Programming).  CORRETA: Conforme texto da Aula 02 no Tema 01  Pair Programming (estilo verificação), é uma forma de nivelar os colaboradores do time de  desenvolvimento que observam em conjunto tanto regras de negócio quanto boas práticas e  padrões de programação. Estilo de verificação intimamente relacionado ao método ágil XP. |
|  | B | SCRUM. |
|  | C | Kanban. |
|  | D | Lean. |
|  | E | FDD. |

Questão 3/5 - Qualidade de Software

A inquietação sobre propostas em modelos de qualidade de software (produto e processos) vem de longa data, anteriores mesmo a 1976. Mas o artigo que marcou o antes e o depois da qualidade de software foi “Avaliação quantitativa da qualidade do software”.

Este artigo foi escrito por Boehm, Brown e Lipow para:

Nota: 20.0

|  |  |  |
| --- | --- | --- |
|  | A | acm.org  **Você acertou!**  CORRETA: Conforme texto da Aula 01 no Tema 03  Mesmo que hajam livros ou artigos anteriores a 1976, vamos focar nosso ponto histórico num  framework para estabelecimento de critérios de qualidade de software sob o artigo intitulado  “AVALIAÇÃO QUANTITATIVA DA QUALIDADE DO SOFTWARE” dos autores Boehm,  Brown e Lipow. [BOEHM, 1976] Este artigo é um clássico da ciência da computação na  plataforma da acm.org. Ele teve por finalidade um estudo para o estabelecimento de uma  estrutura conceitual com alguns resultados iniciais importantes para a análise das características  da qualidade de software. |
|  | B | ieee.org |
|  | C | Modern Computing |
|  | D | Computing & Machine |
|  | E | Modern Software Engineering |

Questão 4/5 - Qualidade de Software

A ISO 12207 é uma norma que certifica sistemas de gestão de qualidade. Esta norma especifica fatores relacionados aos requisitos dentro das atividades de desenvolvimento de software. Ela possui quatro níveis principais: processos fundamentais, processos de apoio, processos organizacionais e processos de adaptação.

Quais são os subníveis do nível processos fundamentais?

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | Identificação do ambiente do projeto, solicitação de informações, seleção de processos,  atividades e tarefas e documentação das decisões e motivos de adaptação. |
|  | B | Documentação, gerência de configuração, gerência de qualidade, processo de verificação,  processo de validação, processo de revisão conjunta, processo de auditoria, processo de  resolução de problemas. |
|  | C | Processo de aquisição, fornecimento, desenvolvimento, operação e manutenção.  CORRETA: Conforme texto da Aula 03 no Tema 2  Vamos detalhar um pouco sobre as funções de cada tipo de Subprocesso dos 4 principais processos  do ciclo de vida da ISO 12207.                Processos Fundamentais são os processos básicos e responsáveis pelo desenvolvimento do  software e subdivididos em:    Processo de aquisição: refere-se a todas as aquisições necessárias para atendimento aos clientes,  desde compras de prateleiras, até ferramentas computacionais.    Processo de fornecimento: refere-se à proposta de revisão e finalização de contrato com  fornecedores.    Processo de desenvolvimento: refere-se a tudo que diz respeito ao código, atividades de  levantamento de requisitos, análise, testes, implantação e aceitação.    Processo de operação: refere-se às atividades entre a utilização do software e o suporte ao usuário.    Processo de manutenção: refere-se às atividades de solução de problemas do produto. |
|  | D | Processo de gerência, de infraestrutura, de melhoria e de treinamento. |
|  | E | Processo de certificação, validação e verificação de erros, defeitos e falhas. |

Questão 5/5 - Qualidade de Software

Padrões de software estão relacionados tanto a produtos (software), quanto a processos (do desenvolvimento do software). Padrões de produtos incluem documentação, estruturação dos requisitos, definição de classes e da codificação do produto.

Enquanto que os padrões de qualidade para processos de desenvolvimento de software incluem:

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | Código limpo, padrões de qualidade total e documentação. |
|  | B | Modelos ágeis, Código limpo, padrões de processo e documentação. |
|  | C | Estruturação dos requisitos, definição de classes e documentação. |
|  | D | Codificação do produto com qualidade, Modelos Ágeis, Código limpo e documentação. |
|  | E | Boas práticas de desenvolvimento, definições de processos e de especificações, validação,  ferramenta de apoio a processos e documentação.  CORRETA: Conforme texto da Aula 01 no Tema 02 Padrões de Processo (para desenvolvermos o software). Produtos que são desenvolvidos com  padrões de qualidade requerem que seus processos também tenham garantia de qualidade. Nos  padrões de processos encontramos boas práticas de desenvolvimento, definições de processos e  de especificações, validação, ferramenta de apoio a processos e documentação. |

Questão 1/5 - Qualidade de Software

As métricas são elementos intrínsecos à qualidade de software, podendo ser algo em relação à documentação ou meta dentro do processo de desenvolvimento de software. Estas métricas abordam situações tais como: linha de código, falhas e erros, por exemplo. Para facilitar nossa compreensão sobre as métricas, elas podem ser divididas em relação ao tempo, aos recursos e às ocorrências.

As métricas em relação aos recursos estão relacionadas:

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | A um determinado tempo que um processo leva para ser concluído. |
|  | B | A um determinado evento como erro, defeito, inspeção de código, número de mudanças nos requisitos e número médio de defeito por linhas de códigos alteradas. |
|  | C | Ao tempo e recurso de um determinado processo. |
|  | D | Ao tempo e evento sobre um determinado processo. |
|  | E | Aos recursos que são utilizados para que um determinado processo seja executado. Como por exemplo, a medição de esforço total de número de pessoas por dia, custos de viagens e alocação de recursos em nuvem.  CORRETA: Conforme texto da Aula 01 no Tema 04, subtema 4.1  Métrica em relação aos recursos necessários para que um processo seja executado. Neste caso, podemos mensurar o esforço total em número de pessoas por dia, custos de viagens e alocação de recursos em nuvem. |

Questão 2/5 - Qualidade de Software

A avaliação da qualidade de software segundo a  ISO/IEC 9126 tem por título geral “Engenharia de software - qualidade do produto”. Esta norma é constituída por 6 subdivisões.

A ISO/IEC 9126 está subdividida em:

Nota: 20.0

|  |  |  |
| --- | --- | --- |
|  | A | Qualidade interna, qualidade externa, processo, engenharia de produto, metodologias e ferramentas |
|  | B | Funcionalidade, engenharia de produto, metodologias, ferramentas, portabilidade e testes. |
|  | C | Funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.  **Você acertou!**  CORRETA: Conforme texto da Aula 01 no Tema 04 |
|  | D | Qualidade externa, qualidade de processo, qualidade de produto, metodologias, boas práticas e ferramentas. |
|  | E | Funcionalidade, confiabilidade, processos, engenharia, qualidade e métricas. |

Questão 3/5 - Qualidade de Software

O controle de qualidade previne vários tipos de defeitos antes que um software seja lançado. São várias técnicas que podem ser utilizadas, dentre as quais encontram-se testes estruturais (tipo caixa-branca), testes funcionais (tipo caixa-preta), testes não funcionais, testes de regressão e revisões.

Quais testes a seguir são do tipo estruturais (caixa-branca)?

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | Testes unitários e testes de integração.   Conforme texto da Aula 02 no Tema 03  Com o controle de qualidade conseguimos prevenir vários defeitos antes do lançamento do produto. Ele encontra-se no estágio final do processo, e para tanto utilizamos algumas ferramentas e ou profissionais especializados para revisar a qualidade do software.  As principais técnicas utilizadas no controle de qualidade encontram-se em testar e revisar. Isto envolve a aplicação de:   * Testes Unitários (teste estrutural - caixa-branca) * Testes de Integração (teste estrutural - caixa-branca) * Testes de Sistema (teste funcional - caixa-preta) * Testes de aceitação (teste funcional - caixa-preta) * Testes não funcionais (usabilidade, carga, segurança, confiabilidade, escalabilidade) * Testes de regressão (manutenção, confirmação) * Revisões de Requisitos e conceitos * Revisões de código * Revisões no Deployment (Implantação) |
|  | B | Testes de sistema e de aceitação. |
|  | C | Testes de usabilidade, carga e segurança. |
|  | D | Testes de confiabilidade e escalabilidade. |
|  | E | Revisões de requisitos, código e deployment. |

Questão 4/5 - Qualidade de Software

DoR (Definition of Ready) é um acordo entre os times de desenvolvimento e o Product Owner, utilizando-se todas as histórias dos usuários para que o backlog esteja cada vez mais refinado. Isto melhora o planejamento e a especificação das cards do backlog.

Uma Card que esteja em modo “Ready” representa:

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | Uma card com atividades entregues pelo time de desenvolvimento. |
|  | B | Uma card com atividades ainda em refinamento. |
|  | C | Uma card com atividades para o time da qualidade trabalhar após a entrega pelo time de desenvolvimento. |
|  | D | Uma card com itens no backlog a ser refinada. |
|  | E | Uma card com atividades para o desenvolvimento que esteja descrita de forma compreensível para que possa ser implementada  CORRETA: Conforme texto da Aula 04 no Tema 1.3  O que é e como funciona a definição de pronto (DoR)?  Nada mais é que um acordo entre os times de desenvolvimento e o Product Owner, utilizando todas as histórias dos usuários para que o Backlog esteja o mais refinado possível. Isto melhora muito o planejamento e a especificação das cards do backlog. Quanto mais refinada uma história, mais cards pequenos e autocontidos. Card? Sim, cada desenvolvedor ou outro profissional que esteja trabalhando no mesmo produto, recebe uma card ou várias cards por Sprint. A Card contém a descrição da atividade que o dev irá fazer. Pode ser uma atividade para o desenvolvimento de uma feature, pode ser uma card para a documentação de uma features, ou quaisquer outras atividades inerentes ao produto. Para que um item do backlog faça parte de uma Sprint, o time deve ter certeza de que ele está “Ready”, ou seja, bem descrito e compreensível para que o desenvolvimento ocorra da forma correta.  É importante que 10% do tempo de cada Sprint seja reservado para o planejamento e adequação da atividade. Isto reduz o impacto de itens mal especificados, os quais geralmente surpreendem de forma ruim o time, e podem gerar mudanças, o que impacta na entrega e no objetivo da Sprint. |

Questão 5/5 - Qualidade de Software

A avaliação da qualidade de software segundo a  ISO/IEC 9126 tem por título geral “Engenharia de software - qualidade do produto”. Esta norma é constituída por 6 subdivisões: Funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

Dentro da Usabilidade encontramos as seguintes características:

Nota: 0.0

|  |  |  |
| --- | --- | --- |
|  | A | Compreensibilidade, apreensibilidade e operacionalidade.  Parabéns ! |
|  | B | Maturidade, apreensibilidade e operacionalidade.  CORRETA: Conforme texto da Aula 01 no Tema 04  Usabilidade: o software é fácil de ser utilizado? Subcaracterísticas:  compreensibilidade: o software tem fácil entendimento sobre o conceito lógico e sua aplicabilidade?  apreensibilidade: é fácil de aprender a usá-lo?  operacionalidade: é fácil operar e controlar? |
|  | C | Compreensibilidade,  processos e qualidade.  CORRETA: Conforme texto da Aula 01 no Tema 04  Usabilidade: o software é fácil de ser utilizado? Subcaracterísticas:  compreensibilidade: o software tem fácil entendimento sobre o conceito lógico e sua aplicabilidade?  apreensibilidade: é fácil de aprender a usá-lo?  operacionalidade: é fácil operar e controlar? |
|  | D | Maturidade, operacionalidade e processos.  CORRETA: Conforme texto da Aula 01 no Tema 04  Usabilidade: o software é fácil de ser utilizado? Subcaracterísticas:  compreensibilidade: o software tem fácil entendimento sobre o conceito lógico e sua aplicabilidade?  apreensibilidade: é fácil de aprender a usá-lo?  operacionalidade: é fácil operar e controlar? |
|  | E | Compreensibilidade, usabilidade e metodologia.  CORRETA: Conforme texto da Aula 01 no Tema 04  Usabilidade: o software é fácil de ser utilizado? Subcaracterísticas:  compreensibilidade: o software tem fácil entendimento sobre o conceito lógico e sua aplicabilidade?  apreensibilidade: é fácil de aprender a usá-lo?  operacionalidade: é fácil operar e controlar? |