QUALIDADE DE SOFTWARE

AULA 2

Profª Maristela Weinfurter

CONVERSA INICIAL

GESTÃO E CULTURA DE QUALIDADE DE SOFTWARE

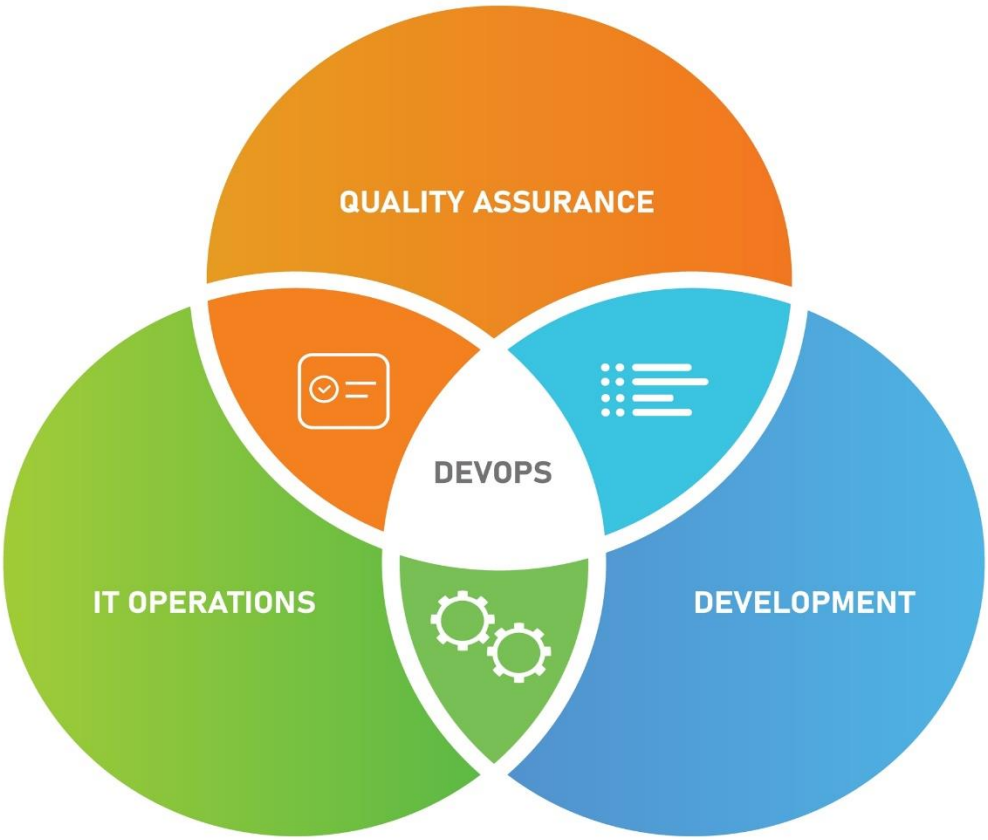
Falar sobre a gestão da garantia da qualidade (Software Quality Assurance – SQA) é um tema que envolve outras siglas, como Software Quality Control (SQC), Software Quality Management (SQM) e Software Processes Improvement (SPI). E todas essas etapas da SQA trazem consigo exatamente os profissionais que atualmente são assinalados nas vagas para tal área, como: analistas de QA, QA Engineer, analistas de testes, especialistas em QA, e assim por diante. Cada um dentro das várias atividades relacionadas à garantia e ao controle da qualidade de software. Esses profissionais não necessariamente precisam ter experiência como desenvolvedores, mas sim terem o conhecimento suficiente para conseguir trabalhar lado a lado com as equipes de desenvolvimento de software.

A Figura 1 ilustra a intersecção existente entre QA e demais áreas da empresa com as quais tais profissionais devem colaborar diariamente.

A SQA envolve processos, atividades de controle de qualidade, utilização de métodos, técnicas e ferramentas, controle de cada ciclo de vida do software, artefatos e códigos, conhecimento de padrões e normas relacionadas à qualidade de software, bem como métricas para que todo o controle realmente seja demonstrado graficamente ou por meio de relatórios para que as devidas ações sejam tomadas sobre a melhoria, a manutenção sejam efetivas sobre o produto (software) e sobre o processo de desenvolvimento de software.

A gestão da garantia da qualidade é essencial para que tenhamos métricas que demonstrem o andamento de cada etapa do processo, bem como seus resultados positivos e negativos. Só é possível a gerência de algo sobre indicadores numéricos. Sem números, a gestão fica extremamente subjetiva e com isso tendemos a não alcançar os resultados necessários sobre o software.

Figura 1 – Software Quality Assurance (SQA)



Crédito: Bakhtiar Zein/Shutterstock.

A engenharia de software, em cada uma de suas épocas, sempre traz um nível de complexidade para o time de qualidade. Hoje, vivemos rodeados de aparelhos que se comunicam, e para toda essa comunicação existir, sempre haverá algum tipo de software envolvido: software embutido em algum objeto (Smart TVs, consoles de videogame, refrigeradores, condicionadores de ar, automóveis, entre outros), software para Web, software para smartphones, tablets, e assim por diante. Na contrapartida, o desenvolvimento também trouxe outros tipos de profissionais para o dia a dia da engenharia de software. Designers de UI, outros profissionais, desenvolvedores, gerentes de produto, analistas de dados, analistas de software, entre outros. A complexidade se multiplicou em termos de hardware, software e infraestrutura, em termos de

linguagens, frameworks, bibliotecas, bancos de dados e integração entre todas essas coisas.

E a complexidade entre recursos tecnológicos e profissionais só aumenta. A cada dia a necessidade de garantir a qualidade do que estamos construindo e de como estamos construindo vai tomando maior importância dentro da área de Tecnologia da Informação.

TEMA 1 – GESTÃO DA QUALIDADE DE SOFTWARE (SQM)

O gerenciamento de qualidade de software (Software Quality Management – SQM) é constituído de uma coleção de processos que garantam que produtos de software, serviços e implementações de processos de ciclo de vida atendam aos objetivos organizacionais de qualidade de software e alcancem a satisfação das partes interessadas.

O SQM é constituído por três subcategorias básicas (Mistrik, 2015) (Figura 2):

1. O planejamento de qualidade de software (Software Quality Planning – SQP);
2. A garantia de qualidade de software (Software Quality Assurance – SQA);
3. O controle de qualidade de software e melhoria de processo de software (Software Quality Control – SQC).
    i. Melhoria do Processo de Software (Software Process Improvement – SPI), este, por sua vez, é visto como uma subcategoria separada, apesar de estar incluída em qualquer uma das três principais categorias anteriormente listadas.

Figura 2 – Subcategorias da SQM

Logo após a introdução dos primeiros computadores e linguagens de programação, o software tornou-se crítico para muitas organizações. O termo “crise de software”, muito difundido na introdução de engenharia de software, foi estabelecido na Conferência de Engenharia de Software da Otan em 1968 (Garmisch, Alemanha). A crise de software geralmente se estabelece de diferentes maneiras, incluindo projetos que excedem os custos estimados para desenvolvimento, atraso na entrega do software e baixa qualidade do software entregue. Mesmo na atualizada, o software continua sendo um elemento crítico em muitas empresas. Para gerenciar os desafios do desenvolvimento de software e garantir a entrega de software de alta qualidade, uma ênfase considerável na comunidade de pesquisa foi direcionada para fornecer subsídio ao gerenciamento da qualidade de software.

Para atingir esses níveis exigentes de qualidade de software, organizações e equipes precisam definir e implementar um rigoroso processo de qualidade de software, enquanto os processos de desenvolvimento precisam incorporar técnicas e ferramentas de garantia de qualidade apropriadas. Isso inclui avaliação de qualidade de requisitos, arquitetura, design e tecnologias de destino, bases de código e ambientes de implantação e tempo de execução (Mistrik, 2015). O teste de software tem sido tradicionalmente um dos pilares da garantia de qualidade, embora muitas outras práticas de gerenciamento de qualidade também sejam necessárias. Os testes se tornaram muito mais

desafiadores com processos de desenvolvimento mais recentes, incluindo métodos ágeis e arquiteturas de serviço mais complicadas e entrelaçadas.

Um software na atualidade incorpora um nível de complexidade bastante grande, tanto a nível de *backend* quanto de *frontend*, ou outros tipos de software. E muito dessa complexidade se deve ao uso de APIs e bibliotecas de terceiros e executados em plataformas de terceiros, com isso, o teste é ainda mais difícil. As várias plataformas de Cloud também podem ter requisitos diferentes e visões diferentes do que é “qualidade” e como deve ser medida e avaliada. Diferentes times de desenvolvimento que colaboram explicitamente em projetos globais de engenharia de software podem ter diferentes práticas de garantia de qualidade, processos de desenvolvimento, arquiteturas, tecnologias, ferramentas de teste e práticas de manutenção.

O Padrão IEEE Std 640.12-1990, segundo Mistrik (2015), estabelece algumas definições importantes para o SQA:

- um padrão que pode ser planejado e sistematizado para ações que fornecem confiança ao software e está em conformidade com requisitos técnicos estabelecidos;
- um conjunto de atividades projetadas para avaliação do processo de desenvolvimento de software;
- atividades planejadas e implementadas dentro do sistema de qualidade para fornecer confiança a quem cumprirá os requisitos de qualidade; e
- parte da gestão de qualidade para garantir que os requisitos serão cumpridos.

O SQP define as metas de qualidade a serem alcançadas, riscos esperados e gerenciamento de riscos, e a estimativa do esforço e cronograma das atividades de qualidade de software. Um SQP geralmente inclui componentes SQA como estão ou personalizados para as necessidades do projeto. Qualquer desvio de um SQP do SQA precisa ser justificado pelo gerente do projeto e confirmado pela gerência da empresa responsável pelo SQA.

As atividades de SQC examinam os artefatos do projeto (por exemplo, código, design e documentação) para determinar se estão em conformidade com os padrões estabelecidos para o projeto, incluindo requisitos e restrições funcionais e não funcionais. O SQC garante, assim, que os artefatos sejam verificados quanto à qualidade antes de serem entregues. Exemplos de atividades de SQC incluem inspeção de código, revisões técnicas e testes.

As atividades de SPI visam melhorar a qualidade do processo, incluindo eficácia e eficiência, com o objetivo final de melhorar a qualidade geral do software. Na prática, um projeto de SPI normalmente começa mapeando os processos existentes da organização para um modelo de processo que é então usado para avaliar os processos existentes. Com base nos resultados da avaliação, um SPI visa alcançar a melhoria do processo. Em geral, a suposição básica para o SPI é que um processo bem definido, por sua vez, terá um impacto positivo na qualidade geral do software.

Uma área muito desafiadora da garantia de qualidade de software (SQA) é a segurança e a privacidade. Sistemas distribuídos de grande escala, com uso intensivo de software, hospedados na nuvem, são inerentemente mais vulneráveis a ataques, perda de dados e outros problemas. As violações de segurança são uma área em que, mesmo que todas as outras preocupações de qualidade com um sistema de software sejam atendidas, problemas massivamente prejudiciais podem resultar de um único e grave problema de segurança.

TEMA 2 – GARANTIA DA QUALIDADE DE SOFTWARE (SQA)

Apesar de que na atualidade se fala exaustivamente de modelos ágeis, o SQA foi introduzido em conjunto, no passado, com vários tipos de metodologias de desenvolvimento de software, como: cascata, prototipagem, desenvolvimento iterativo e incremental, desenvolvimento em espiral e desenvolvimento rápido de aplicativos. Os defensores do paradigma de desenvolvimento ágil de software argumentam que, para qualquer projeto não trivial, terminar uma fase do ciclo de vida de um produto de software perfeitamente antes de passar para as próximas fases é praticamente impossível.

2.1 SQA e modelos de melhoria de processo

Um processo maduro e bem definido colabora com o desenvolvimento de software de qualidade com um número substancialmente reduzido de defeitos. Alguns exemplos populares de modelos de melhoria de processo incluem o Capability Maturity Model Integration (CMMI) do Software Engineering Institute, ISO/IEC 12207 e Software Process Improvement and Capability Determination (SPICE).

O Capability Maturity Model Integration (CMMI) é um modelo composto por práticas importantes para maturidade de disciplinas específicas, como engenharia de software. Esse modelo é administrado pelo Instituto CMMI com base em melhores práticas para o desenvolvimento e manutenção de software, dividido em cinco níveis de maturidade. O CMMI foi sendo construído desde a década de 1980, tendo sido o ano de 1991 o marco para o desenvolvimento das CMMs, as quais antecedem o CMMI. A constituição do CMMI tem por base o SW-CMM (SEI Software CMM), EIA SECM (Electronic Industries Alliances's Systems Engineer Capability Model) e IPD-CMM (Integrated Product Development CMM).

Os níveis de maturidade estabelecidos no CMMI são os que se seguem.

- **Nível 0: Incompleto** – os processos não funcionam ou não atingem todas as metas e objetivos definidos pela CMMI.
- **Nível 1: Executado** – os processos definidos pela CMMI já estão sendo executados com tarefas que produzem artefatos definidos.
- **Nível 2: Controlada** – 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 de que as tarefas e o software são monitorados, controlados e revisados em conformidade com os processos da CMMI para esse nível.
- **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.
- **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.
- **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 melhoria contínua do processo.

A implantação da CMMI leva uma empresa a conquistar um processo de desenvolvimento maduro de forma iterativa. À medida que cada nível é atingido, os processos ficam mais consistentes e a gestão da qualidade consegue melhorar sensivelmente no número de bugs. Não podemos esquecer de que quaisquer modelos de maturidade para o ciclo de vida do software geram custos a curto prazo que os reduzem de novos projetos a médio e longo prazo, pois com o número de problemas caindo, a manutenção corretiva também diminui, liberando o time de desenvolvimento para a implementação de novas features e não de correções daquilo que já deveria estar funcionando.

2.2 SQA e padrões de software

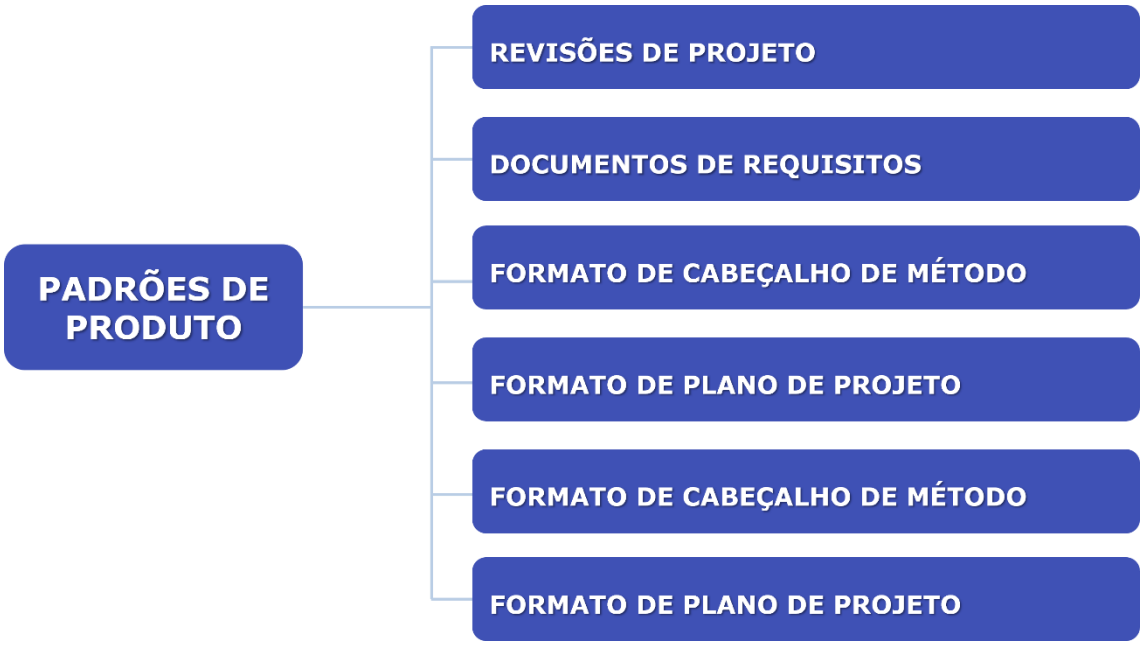
Padrões de projeto de software são soluções genéricas para problemas recorrentes. A qualidade do software pode ser suportada pela reutilização de padrões de projeto que foram comprovados no passado.

Padrões de software são importantes porque se baseiam no conhecimento sobre a forma mais adequada e prática para as empresas, bem como auxiliam na reutilização da experiência para que erros do passado sejam evitados. Como qualidade é algo subjetivo, a utilização de padrões estabelece a base para 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 feita de forma tranquila, causando menos esforço de aprendizagem para que outros profissionais possam dar prosseguimento às atividades de desenvolvimento do software.

Padrões são trabalhados em duas dimensões: produto e processo:

- Padrões de Produto (produto = software). Desde 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 3 trata de alguns exemplos 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 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. A Figura 4 exemplifica padrões relacionados ao processo de desenvolvimento de software.

Figura 4 – Exemplos de padrões de processo

Garantir a qualidade de software durante o processo de desenvolvimento de software envolve várias questões, especialmente os modelos de maturidade e os padrões como vistos até aqui.

2.3 Garantindo a Qualidade de Software (SQA)

Uma área muito desafiadora da garantia de qualidade de software (SQA) é a segurança e a privacidade. Sistemas distribuídos de grande escala, com uso intensivo de software, hospedados na nuvem, são inerentemente mais vulneráveis a ataques, perda de dados e outros problemas. As violações de segurança são uma área em que, mesmo que todas as outras preocupações de qualidade com um sistema de software sejam atendidas, problemas massivamente prejudiciais podem resultar de um único e grave problema de segurança.

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 essas 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-se os 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 e são falhas de qualidade de software quase que inevitáveis.

A garantia de qualidade de software precisa levar em consideração alguns aspectos importantes como:

- foco no cliente;
- boa gestão;
- envolvimento de técnicos e usuários;
- processos bem claros;
- gestão dos processos;
- melhoria contínua;
- decisões baseadas em fatos e informação; e
- 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, por meio 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 finalidade dele, pode gerar consequências mais brandas ou mais catastróficas.
- **administração de riscos** – todo projeto envolve riscos, e eles devem ser previstos por meio de planos de contingência.

SQA é um trabalho exigente e demorado. Mas é um trabalho essencial para que os desenvolvedores produzam software que estejam em conformidade com suas especificações e atendam às expectativas do cliente. O SQA é composto pelas seguintes atividades/elementos (Laporte, 2018).

- **processo** – é fundamental definir e documentar um processo robusto e endossado por especialistas do setor ao tentar criar uma cultura de qualidade em qualquer organização.
- **compromisso da gestão** – a organização deve estar disposta a comprometer os recursos necessários (tempo, pessoas, ferramentas, equipamentos) para cumprir as metas de qualidade para produtos em desenvolvimento e para SPI.
- **experiência pessoal** – as habilidades do pessoal do projeto geralmente determinam o sucesso de um projeto, e o trabalho de qualidade de software não é exceção.
- **produtos** – requisitos, planos de teste e casos de teste devem ser definidos no início do processo de desenvolvimento para que os desenvolvedores criem produtos de software a partir de uma perspectiva de qualidade.
- **uso de ferramentas** – o uso de ferramentas para rastrear e gerenciar defeitos, bem como criar e executar casos de teste é necessário para aumentar a maturidade do processo.
- **métricas** – o desenvolvimento de métricas de qualidade para rastrear a qualidade, permitindo a comparação com as metas de qualidade, aumentará o valor e a maturidade dos processos de V&V.
- **ambiente de teste** – deve permitir que os desenvolvedores reproduzam condições em ambientes de produção ao processar casos de teste.
- **dados de teste** – os dados devem estar disponíveis para os desenvolvedores quando a programação do processo exigir a execução dos casos de teste de um componente.
- **gerenciamento de alterações** – os processos de teste e desenvolvimento devem rastrear as alterações de configuração e garantir que os produtos sejam executados em ambientes de produção.
- **conscientização do desenvolvedor** – todo profissional de negócios e todo desenvolvedor deve estar ciente do processo de qualidade de software e acreditar que ele agrega valor ao seu trabalho diário.

A gestão da qualidade de software deve levar em consideração todas as possibilidades de problemas que usualmente ocorrem no software. Só identificando as reais necessidades dos usuários aliadas às experiências com problemas no desenvolvimento de software conseguiremos compreender e aplicar os conceitos, modelos e padrões de software adequados a cada projeto de software.

TEMA 3 – CONTROLE DA QUALIDADE (SQC)

O controle de qualidade de software está relacionado a atividades de avaliação da qualidade do produto final de software. Seus principais objetivos são impedir que o software com problemas se qualifique e minimize o custo a fim de garantir a qualidade com uma variedade de atividades de infraestrutura e atividades realizadas ao longo dos processos de desenvolvimento de software.

Atividades relativas à SQC previnem as causas dos erros e detectam e corrigem 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, e reduzirá 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.

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 nele.

O controle de qualidade previne contra vários defeitos antes do lançamento do produto. Ele se encontra no estágio final do processo; para tanto, utilizamos algumas ferramentas e/ou profissionais especializados para revisar a qualidade do software.

As principais técnicas utilizadas no controle de qualidade são de testes e revisões:

- 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; e
- 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 tem por finalidade básica testar individualmente o código. Nesse 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. E 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 deve 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, temos o que se segue.

- **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, 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 fazem parte dos testes iniciais do código, e deve ser o mais atômico possível, 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 fazemos 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 por ferramentas. Essas *open-source* ou não, facilitam e agilizam esse 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 os mesmos 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 esses testes não estão associados a funcionalidades, mas a 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 no 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.

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 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 esse 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 desse 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. Essa tarefa pode ser efetuada por uma equipe ou então por um desenvolvedor mais experiente. O importante nesse 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.

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 contínua, garantindo que a implantação do novo software ou das alterações efetuadas ocorram de forma rápida, segura e correta. Essas 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 revisões e *deployment*. O processo automatizado gera relatórios de quebras da build quando ocorrem falhas. Falhas que podem ser *commits* esquecidos, versões de bibliotecas de códigos, entre outras situações.

Técnicas e revisões são as principais características do controle de software. Ao serem implantadas dentro do ciclo de vida do produto de software, contribuem para que cada fase posterior o produto já esteja mais depurado e contendo menos erros. Então, quanto mais cedo houver a implantação de alguma técnica de controle de qualidade, como no caso testes unitários, menos custos teremos com manutenções.

TEMA 4 – PLANEJAMENTO DA QUALIDADE DE SOFTWARE (SQP)

O Planejamento de Qualidade de Software Software Quality Planning (SQP) é definido a nível do projeto e está alinhado com a Garantia da Qualidade de Software (SQA). Ele especifica o compromisso do projeto de seguir o conjunto de normas, regulamentos, procedimentos e ferramentas aplicáveis e selecionados durante o ciclo de vida do desenvolvimento.

Além disso, o SQP define as metas de qualidade a serem alcançadas, os riscos esperados e o gerenciamento de riscos, e a estimativa do esforço e cronograma das atividades de qualidade de software. Um SQP geralmente inclui componentes da SQA para as necessidades do projeto. Qualquer desvio de um plano de projeto de qualidade da garantia de qualidade precisa ser justificado pelo gerente do projeto e confirmado pela gerência da empresa responsável pelo SQA.

Às vezes, é difícil fazer com que todas as partes interessadas concordem

se um sistema apresenta alta qualidade ou não, se seus julgamentos dependem inteiramente da opinião pessoal. Para tanto, é importante a adoção de métricas para que possamos mensurar a qualidade do software. Para que tudo funcione adequadamente, é importante que os desenvolvedores colaborem no momento de relatar os defeitos para que o pessoal de qualidade possa coletá-los e transportá-los para gráficos e relatórios de qualidade. Muitas vezes, tais dados de defeitos podem estar incompletos, incorretos ou imprecisos em relação às causas de defeitos específicos. Isso poderá comprometer a melhoria do processo SPI, pois dados incompletos causarão desvios nas medições.

Um dos pontos importantes do SQP é então produzir, coletar e validar evidências para provar que o produto de software atenda aos requisitos funcionais e não funcionais exigidos.

Para atingir uma boa garantia pelo planejamento da garantia da qualidade, devemos:

- identificar as normas e os procedimentos necessários;
- descrever como as medidas e atributos escolhidos representam adequadamente a qualidade do produto;
- utilizar essas medidas e identificar lacunas entre objetivos e resultados; e
- garantir a qualidade dos procedimentos de medição do produto e a eficiência ao longo do projeto.

Há uma norma (IEEE 730) que recomenda que um plano SQA deve definir o conceito de qualidade do produto para melhoria contínua (SPI). Esse documento deve abordar as questões fundamentais de funcionalidade, interfaces externas, desempenho, características de qualidade e restrições computacionais impostas por uma implementação específica. Cada requisito deve ser identificado e definido para que sua implementação possa ser validada e verificada objetivamente.

Ao elaborar nosso planejamento de qualidade de software, é importante observar dois fatores:

1. Fatores para o aumento da qualidade do software, que aborda:
    a. boa compreensão da qualidade não funcional;
    b. bom processo para definir, acompanhar e comunicar requisitos de qualidade;
    c. avaliação da qualidade ao longo do ciclo de vida do software;

d. estabelecer a criticidade do software antes de iniciar um projeto; e
e. usar os benefícios da rastreabilidade de software.

2. Fatores que afetam a qualidade do software, os quais são:
   a. não levando em consideração os requisitos de qualidade;
   b. não levando em consideração a criticidade do software; e
   c. arrumar desculpas para não se preocupar com a qualidade.

O plano de garantia da qualidade de software faz referência aos artefatos e documentos utilizados no planejamento do ciclo de vida do software.

Por exemplo, como definições, acrônimos e abreviações, pode-se referenciar ao Glossário estabelecido pelo time de desenvolvimento. Ferramentas, técnicas e metodologias fazem referência ao plano de métodos, ferramentas e técnicas do desenvolvimento, e assim por diante.

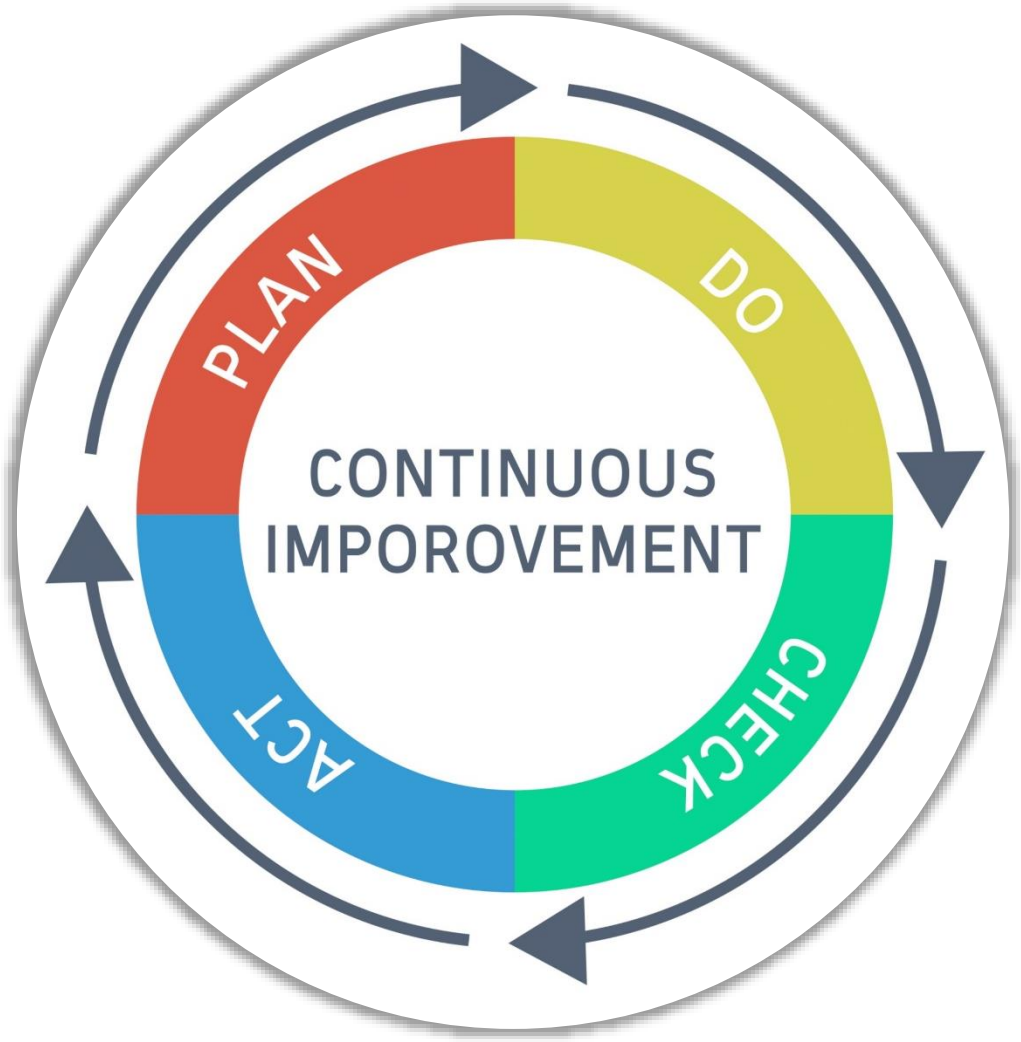
Em síntese, para que um plano de garantia da qualidade de um produto de software ocorra, é necessário que ele se baseie exatamente sobre o planejamento da área de desenvolvimento.

Somente ao compreendermos como os artefatos são gerados somos capazes de montar o melhor plano de garantia de qualidade, envolvendo verificações, validações, testes, auditorias, entre outras atividades.

TEMA 5 – SPI – MELHORIA CONTÍNUA, MODELOS E NORMAS

A Melhoria Contínua é um conceito muito utilizado em vários tipos de negócio e com o desenvolvimento de software não é diferente. A Gestão de Qualidade Total (Total Quality Managment – TQM), idealizada por Deming, utiliza o ciclo de melhoria contínua chamado PDCA (*plan, do, check, action*), que pode ser visualizado na Figura 5.

Figura 5 – Melhoria contínua utilizando PDCA



Crédito: Hermanthos/Shutterstock.

O Ciclo PDCA é um dos métodos mais conhecidos e aplicados especialmente dentro das indústrias. PDCA significa o que se segue.

- Plan (Planejar) – são as expectativas e os planos que a empresa tem em relação ao processo de fabricação ou, no nosso caso, de desenvolvimento de software. A meta é atingir 100% de sucesso na produção ou execução de serviços.
- Do (Fazer) – momento da implementação do planejado no item 1 do ciclo PDCA. Nesse momento, coleta-se e mapeia dados para análise e elaboração de indicadores que determinarão como estão nossos processos.
- Check (Verificar) – os indicadores estabelecidos no item 2 do ciclo PDCA agora viram objetos de estudo. Nesse momento, são desenvolvidos

métodos para o controle mais eficiente do desenvolvimento de software ou produção e execução de serviços.

- Act (Agir) – após todos os itens anteriores do ciclo do PDCA, o projeto inicial então passa por melhorias, aprimorando-se o nosso processo de desenvolvimento de software ou produção.

Junto às técnicas e ferramentas da gestão da qualidade total, encontram-se as normas e os modelos de maturidade do software, como ISO, ABNT, CMMI e MR-MPS. As normas são elaboradas para que processos e produtos possam passar por processos de certificação, garantindo aos clientes que os produtos e os serviços que eles estão consumindo estejam em conformidade.

Empresas de grande porte e já consolidadas geralmente possuem processos, métodos e técnicas estabelecidas e maduras para o desenvolvimento e garantia da qualidade de software. Mas quando trocamos o foco para as pequenas empresas e iniciantes, como o caso das startups, processos são muito simplificados e por vezes falhos, pois garantir a qualidade, passar por certificações, adotar técnicas, modelos e ferramentas custam, e por vezes é inviável a adoção.

Outra ferramenta bastante difundida dentro da área de qualidade é o Controle Estatístico de Processo (CEP), muito utilizada e difundida no setor de indústria, inclusive dando muitas vezes nome a áreas específicas de acompanhamento e controle do processo fabril. É possível adotar-se o CEP para melhoria do processo de qualidade de software.

Além do CEP, o Seis Sigma (SS) também é um recurso adotado pela área de qualidade e pode se ajustar aos processos de desenvolvimento de software. O SS tem por estratégia identificar, classificar e priorizar as ações de melhoria e inovação para que a empresa atinja seu grau de maturidade no processo.

Agora, 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. Startups iniciais geralmente contam com um quadro reduzido de desenvolvedores e sem profissionais focados apenas em gestão da qualidade. Até porque um dos grandes objetivos das startups focadas em desenvolvimento de software encontra-se sempre no que chamamos de Minimum Viable Product (MVP), ou seja, um pequeno protótipo funcional para que o mercado comece a conhecer sua ideia por meio de um software que ainda é muito simplificado. E

tal MVP serve para divulgar a ideia e o projeto da startup, com o intuito de conquistar investidores e clientes iniciais. Conforme o protótipo inicial vai adquirindo novas funcionalidades e características, também vai adquirindo problemas, ou seja, erros. Nesse momento é que a melhoria contínua pode ser implementada para que a experiência dos usuários obtenha melhores resultados.

As atividades de Software Process Improvement (SPI) visam melhorar a qualidade do processo do desenvolvimento de software, incluindo eficácia e eficiência, com o objetivo final de melhorar a qualidade geral do software. Na prática, um projeto de SPI normalmente começa pelo mapeamento dos processos existentes da organização para um modelo de processo que é então usado para avaliar os processos existentes. Com base nos resultados da avaliação, um SPI visa alcançar a melhoria do processo. Em geral, a suposição básica para o SPI é que um processo bem definido, que, por sua vez, terá um impacto positivo na qualidade geral do software.

O processo de desenvolvimento de software deve ser visto como um processo de qualquer outro tipo de negócio, o qual pode passar por sucessivas melhorias por meio da adoção de metodologias e ferramentas que possam criar maturidade ao desenvolvimento de software. Inclusive, criando métricas que possam auxiliar no gerenciamento dos processos.

5.1 Normas relacionadas à qualidade de processos e software

Para que o processo de desenvolvimento crie maturidade, várias empresas, geralmente de grande porte, adotam a certificação CMMI-DEV, a qual garante que processos entrem dentro de um espiral de melhoria contínua. Já para empresas pequenas e médias, há o MR-MPS.br, a qual segue as mesmas ideias da CMMI, porém por meio um consórcio de empresas, faz com que os custos de implantação e certificação sejam divididos para garantir que as elas possam ter seus processos de desenvolvimento de software mais maduros.

Além do CMMI e do MR.MPS, a seguir verificamos várias normas e boas práticas que se complementam para auxiliar na maturidade dos processos de desenvolvimento de software.

Padrões Internacionais de Processo de Software: ISO/IEC 12207 (norma para a qualidade do processo e desenvolvimento de software).

Padrões Internacionais de Qualidade de Software

- ISO/IEC 9126 (características de qualidade de produto de software, versão brasileira NBR 13596).
- ISO 14598 (conjunto de guias para avaliação de produtos de software com base na norma ISO 9126).
- ISO 12119 (características de qualidade de pacote de software vendido comercialmente).
- IEEE P1061 (metodologia de métricas para padrão de qualidade de software).
- ISO/IEC 15504.
- ISO/IEC 25010 (modelos de qualidade para produtos de software).

Boas práticas: PSP, TSP, ITIL, COBIT.

Modelos de processo de software: modelos SW-CMM, CMMI e MPS.BR.

Outros modelos

- ISO/IEC 29110.
- ISO/IEC 14598.
- IEEE 1074.
- IEEE 1298.

A melhoria contínua traz consigo um conjunto de conceitos, ideias, ferramentas, técnicas e modelos que, ao serem adaptados à realidade de cada empresa, é capaz de conduzir o desenvolvimento de software a um patamar com mais qualidade tanto a nível de processo quanto a nível de produto.

FINALIZANDO

Nossa visão sobre a garantia da qualidade de software deve ser construída com base naquilo que possuímos como estrutura para o desenvolvimento de software. Não necessariamente precisamos adotar exatamente todas as etapas da SQA dentro de nossas empresas. O que precisamos é nos atentar para o que queremos e conseguir trabalhar em termos de qualidade de software e de processo.

A SQM é a coleção de todos os processos que garantem que produtos de software, serviços e implementações de processos de ciclo de vida atendam aos objetivos organizacionais de qualidade de software e alcancem a satisfação das partes interessadas. Descrevemos brevemente as abordagens básicas de SQM,

incluindo SQP, SQA, SQC e SPI. Considerando o tema SQM, nesse contexto, discutimos a curta história e evolução dos modelos de qualidade de software. Além disso, foram discutidas as abordagens atuais de última geração para avaliar as abordagens de qualidade do sistema. Embora exista um grande corpo de conhecimento sobre SQA, ainda existem muitos grandes desafios que requerem nossa atenção.

Quando estamos iniciando com a garantia de qualidade em nossas organizações, o essencial é que foquemos no Controle da Qualidade, com a introdução de testes, verificações e validações. Conseguiremos evoluir para as demais etapas da SQA à medida que adotarmos normas, padrões e atingir o maior grau de maturidade possível, talvez por meio de um CMMI ou de um MPS.br. Tudo depende do tamanho da empresa e de quanto os níveis hierárquicos superiores desejam investir.

Uma estrutura organizacional para suprir todas as áreas da SQA é algo caro, e deve-se levar em consideração o custo-benefício entre o que efetivamente trará resultados que garantam ao final um produto com o menor número de problemas possível.

Há uma variedade enorme de técnicas e ferramentas que podemos adotar. O ideal é sempre dirigir nossa área de qualidade em função de tudo aquilo que foi planejado para o ciclo de vida de cada projeto de software.

O importante que podemos levar desse assunto de SQA é que devemos iniciar bem fundamentados, com poucas técnicas e ferramentas e ir trabalhando com foco na melhoria contínua de nossos processos.

REFERÊNCIAS

ACM. **Site ACM**. 2022. Disponível em: <https://www.acm.org/>. Acesso em: 5 ago. 2022.

ARNON, A. **Complete guide to test automation**: techniques, practices, and patterns for building and maintaining effective software projects. Apress, 2018.

GALIN, D. **Software quality**: concepts and practice. IEEE Press/Wiley, 2018.

IBM. **IBM 2022**. Disponível em: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/compsci/>. Acesso em: 5 ago. 2022.

IEEE. **SWEBOK**. 2022. Disponível em: <https://www.computer.org/education/bodies-of-knowledge/software-engineering>. Acesso em: 5 ago. 2022.

_____. **Site IEEE**. 2022. Disponível em: <https://www.ieee.org/>. Acesso em: 5 ago. 2022.

_____. **IEEE1012**. 2016. Disponível em: <https://ieeexplore.ieee.org/document/8055462>. Acesso em: 5 ago. 2022.

ISO. **ISO 8402**. Disponível em: <https://www.iso.org/standard/20115.html>. Acesso em: 5 ago. 2022.

_____. **ISO 9000**. Disponível em: <https://www.iso.org/iso-9001-quality-management.html>. Acesso em: 5 ago. 2022.

_____. **ISO 24765**. Disponível em: <https://www.iso.org/standard/71952.html>. Acesso em: 5 ago. 2022.

_____. **ISO 25010**. Disponível em: <https://www.iso.org/standard/35733.html>. Acesso em: 5 ago. 2022.

ISTQB. **ISTQB 2011**. Disponível em: <https://www.istqb.org/certifications/certified-tester-foundation-level>. Acesso em: 5 ago. 2022.

LAPORTE, C.; APRIL, A. **Software quality assurance**. IEEE Press. Wiley, 2018.

LÉLIS, E. C. **Gestão da qualidade**. São Paulo: Pearson Prentice Hall, 2012.

MISTRIK, I. **I and Others**. Software Quality Assurance. Elsevier. O'Reilly, 2015.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

SHIGUNOV NETO, A. **Introdução à gestão da qualidade e produtividade**: conceitos, história e ferramentas. Curitiba: InterSaberes, 2016.

SOMMERVILLE, I. **Engenharia de software**. São Paulo: Pearson Education do Brasil, 2018.